프로그람개발문고

Qt프로그람개발도구

교육위원회 교육정보쎈터

주체99(2010)

차 례

머 리 말

위대한 령도자 **김정일**동지께서는 다음과 같이 지적하시였다.

《프로그람을 개발하는데서 기본은 우리 식의 프로그람을 개발하는것입니다. 우리는 우리 식의 프로그람을 개발하는 방향으로 나가야 합니다.》

(《**김정일**선집》제15권, 196페지)

오늘 우리 나라에서는 위대한 령도자 **김정일**동지의 현명한 령도에 의하여 프로그람기술이 빠른 속도로 발전하고있다.

우리의 과학자, 연구사들은 우리 식 조작체계 《붉은별》을 개발하였으며 각종 도구들과 응용프로그람들을 개발하기 위한 연구사업을 활발히 진행하고있다.

Qt는 우리 식 조작체계 《붉은별》과 함께 Linux, Unix, Windows, MAC OS X와 같은 여러 가동환경에서 동작하는 GUI프로그람을 개발하기 위한 C++에 기초하고있는 한조의 도구이다.

Qt는 프로그람의 개발을 쉽게 하고 고속화하기 위한 몇가지 지령행도구들과 도형방식도구들을 제공한다. Qt가 제공하는 도구들은 다음과 같다.

- Qt Designer - 시각적으로 폼을 설계한다.
- Qt Linguist, lupdate와 lrelease - 다국어의 요구에 맞게 응용프로그람을 번역한다.
- Qt Assistant - 필요한 방조를 고속으로 검색한다.
- qmake - 단순한 가동환경에 의존하지 않는 프로젝트파일로부터 Makefile를 창조한다.
- qembed - 자료 실례로 화상들을 C++코드로 변환한다.
- qvfb - 탁상우에서 매몰형 응용프로그람들을 실행하고 시험한다.
- makeqpf - 매몰형장치용의 미리 묘사된 서체들을 창조한다.
- moc - 메타객체콤파일러
- uic - 사용자대면부콤파일러
- qtconfig - 직결방조를 갖추고있는 Unix에 기초한 Qt환경구성도구

이 책에서는 이상과 같이 Qt에 포함되여있는 각종 도구들의 사용방법을 서술한다.

우리는 Qt의 일반원리와 프로그람작성법을 습득함으로써 우리 식의 조작체계에서 실행할수 있는 프로그람들을 더 많이 개발함으로써 나라의 프로그람기술을 한계단 더 발전시키고 인민경제의 정보화를 실현하는데 적극 이바지하여야 한다.

제1장. Qt Designer

이 장에서는 Qt다중가동환경 GUI개발도구와 함께 구축된 사용자대면부의 설계 및 실현도구인 Qt Designer를 설명한다. Qt Designer는 사용자대면부설계에 대한 실험을 쉽게 해준다. Qt Designer가 생성하는 파일들로부터 자기의 요구대로 설계를 변경하여 사용자대면부를 복사하는데 필요한 코드를 생성할수 있다. 낡은 판을 사용하였어도 대면부가 아주 류사하므로 새 판으로 즉시 만들수 있다. 그리고 새로운 창문부품과 새로 개선된 기능을 찾을수 있다.

Qt Designer는 실행시에 자동적으로 자기의 창문부품(Windows용어로 조종요소)들을 이동하거나 비례를 조절할수 있는 배치도구를 사용하여 사용자대면부를 만들수 있다. 결과로 생기는 대면부는 사용자의 조작환경과 애호에 가장 적합하다. Qt Designer는 창문부품들사이의 사건구동통신을 위한 Qt의 신호-처리부기구를 지원한다. Qt Designer는 생성된 코드안에 자기의 전용처리부를 매몰하는데 사용할수 있는 코드편집기를 포함하고있다. 생성된 코드를 손으로 작성한 코드와 분리하는것을 좋아하는 사람들은 초판의 Qt Designer에서 개척된 파생클라스작성수법을 계속 리용할수 있다.

이 장에서는 실례응용프로그람들의 개발을 통하여 Qt Designer를 소개한다. 처음 8개 절들에서는 Qt를 시작하는 방법과 기본응용프로그람, 대화칸, 사용자정의창문부품, 자료기지프로그람의 작성방법, Qt Designer의 전용화와 통합방법을 설명한다.

나머지 절들에서는 Qt Designer의 차림표선택, 도구띠, 지름건, 대화칸, 위자드 및 창문대화칸, 위자드 및 창문에 대하여 설명한다.

1. 무엇을 알아야 하는가

이 책은 C++와 Qt GUI도구일식의 기초지식을 가지고있는것을 전제로 하고있다. Qt의 Enterprise판에는 Qt SQL모듈이 포함되여있다. 자료기지응용프로그람작성에서 Qt Designer를 리용한 SQL응용프로그람건설방법에 대하여 보여준다. 이 절은 SQL과 관계형자료기지에 대한 지식을 요구한다.

2. Qt 3.0용 Qt Designer에서 새로운것은 무엇인가?

3.0판의 Qt Designer는 그 이전판보다 많은 기능을 가지고있다. 실례로 전용처리부용코드를 Qt Designer에서 직접 편집할수 있고 작용, 도구띠, 차림표를 가지는 기본창문을 작성할수 있으며 분할기들을 결합하는 배치를 사용할수 있으며 플라그인은 임의의 개수의 사용자정의창문부품들을 묶어서 그것들을 Qt Designer에 쓸모있게 만들수 있다. 그밖에 사용자대면부의 자그마한 개량으로부터 효과성개선에 이르기까지 다른 많은 기능이 강화되였다.

3.0판의 Qt Designer는 응용프로그람에 있는 모든 폼들사이의 절환과 자료기지

설정과 화상의 공통모임의 관리를 쉽게 하는 프로젝트파일을 도입한다. 파생클라스작성이 완전히 유지되여도 Qt Designer에서 코드의 직접쓰기는 4절에서 학습한다. 또한 새로운 서고 libqui가 도입되여 Qt Designer의 .ui파일들로부터 실행시에 대화칸을 동적으로 적재할수 있다. 이것은 응용프로그람의 사용자들에게 C++를 사용하지 않고도 상당한 대면부전용화능력을 제공하게 한다.

3.0판의 Qt Designer가 새 수법들과 기술을 도입하였다. 단일대화칸설계도구가 요구된다면 이것을 무시하고 Qt 2.x에서 제공된 판을 사용할 때와 꼭같은 방법으로 간단히 사용할수 있다.

1) Qt 3.1용Qt Designer에서 새로운것은 무엇인가?

• 현재 Qt Designer는 최근에 사용한 파일들의 고속호출을 위한 기동대화칸을 제공한다. (필요없다면 그 기능을 차단할수 있다.)

• 신호-처리부대화칸은 변경되였다. 한번의 동작으로 많은 련결을 아주 쉽게 더 빨리 작성할수 있다. 아직은 생성하려는 련결을 선택하고 끌기해야 하지만 새 대화칸은 훨씬 고속이다.

• 창문부품들은 현재 Toolbox에 의하여 호출할수 있다. (모든 원시도구띠들을 여전히 쓸수 있다.) 이것은 도구띠단추와 같은 간단한 호출을 제공하며 창문부품의 이름들을 보여주고 더 적은 공간을 차지한다.

• QWidgetStack는 현재 용기창문부품으로서 사용할수 있다. 새로운 실례는 그 사용법을 보여준다.

• .ui.h파일들은 현재 const정의, #include, 일반함수 등 임의의 C++원천을 포함할수 있다.

• 여러 창문부품들을 선택할 때 그것들의 공통속성들은 속성편집기에 표시되고 집체적으로 변경할수 있다.

2) Qt 3.2용 Qt Designer에서 새로운것은 무엇인가?

• Qt 3.1로부터 Qt Designer에 의해 사용된 도구칸창문부품은 현재 자기의 프로그람에서 사용하기 위한 Qt창문부품으로서 사용할수 있다. 그 창문부품을 QToolBox라고 부른다.

• Qt Designer의 차림표편집기는 다시 설계되여 응용프로그람의 기본창문들을 시각적으로 간단히 만들수 있다.

• 사용가능성과 련관된 소규모의 개량은 3.0판에 아주 많다.

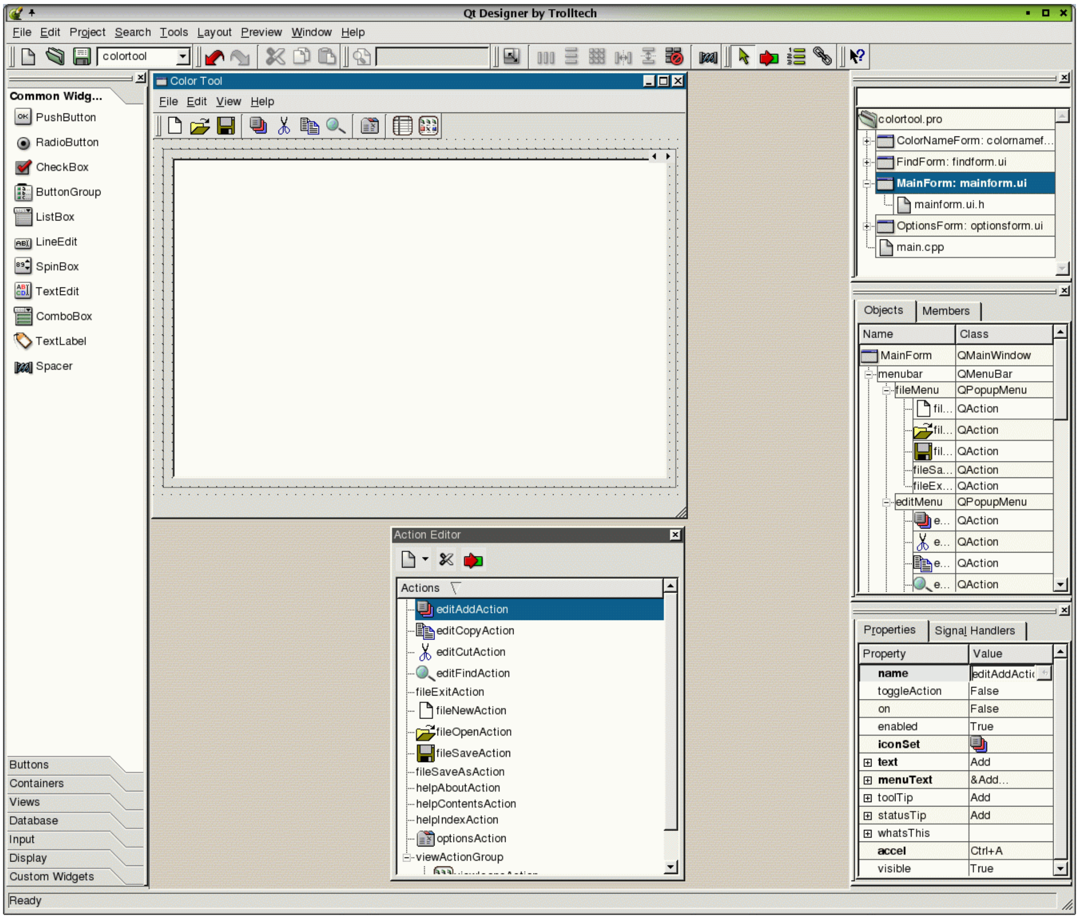


그림 1-1. Qt Designer

제1절. 시작

이 절에서는 사용자들이 Qt Designer를 빨리 시작하기 위한 차림표를 제공한다. 이 절에서는 간단한 대화칸형식의 치수변환프로그람의 작성과정을 하나하나 설명한다. 여기서는 폼에 창문부품의 추가, 창문부품속성의 설정, 신호와 처리부의 련결, 창문부품 배치, 그리고 전용코드의 추가에 대하여 설명한다. 이 절에서는 Qt Designer의 기능과 설명의 일부만 학습하며 구체적인것은 2절과 3절에서 소개하는 colortool프로그람에서 학습한다.

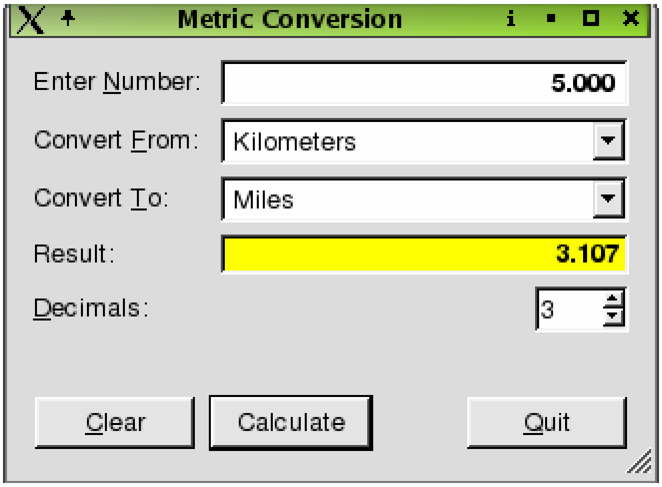


그림 1-2. Metric Conversion대화칸

1. Qt Designer의 기동과 완료

1) Windows에서 Qt Designer의 기동

Windows에서 Qt Designer를 기동하려면 Start단추를 찰칵하고 Programs|Qt X.x.x|Designer.를 찰칵한다. (X.x.x는 Qt판번호 례하면 3.3.1.)

2) Unix 혹은 Linux에서 Qt Designer의 기동

Qt Designer는 사용하고있는 탁상환경에 따라서 여러가지 방법으로 기동할수 있다.

- 탁상배경우에 Qt Designer그림기호가 있으면 그것을 두번 련속 찰칵하여 Qt Designer를 기동할수 있다.
- Start차림표가 있는 탁상환경에서 련관된 보조차림표에 Qt Designer항목이 있으면 Programming 또는 Development보조차림표에서 Qt Designer항목을 찾아서 선택한다.
- 말단창문에서 designer라고 입력하여 Qt Designer를 기동할수도 있다.

3) Mac OS X에서 Qt Designer의 기동

Finder안의 Qt Designer를 련속 두번 찰칵한다.

4) 완료

Qt Designer를 완료하려면 File|Exit를 찰칵한다. 변경을 보관하지 않은 경우에 그것을 보관하겠는가를 묻는 재촉문이 나타난다.

2. 프로젝트의 작성

그러면 Qt Designer가 기동된다. Qt Designer가 기동하면 New/Open대화칸이 열린다. 이 대화칸을 자체로 열수 있으므로 Cancel을 찰칵하여 그것을 닫는다.

치수변환응용프로그람은 표준C++응용프로그람이므로 C++프로젝트를 만들고 여기에 파일들과 코드를 추가하여야 한다.

새 프로젝트를 다음과 같이 만든다.

- File|New를 찰칵하여 New File대화칸을 표시한다.
- C++ Project를 찰칵하여 C++프로젝트를 작성한 다음 OK를 찰칵하여 Project Settings대화칸을 펼친다.
- Project File행편집칸 다음의 생략(...)단추를 찰칵하여 Save As대화칸을 연다. 이 대화칸을 사용하여 새로운 프로젝트를 만들려는 위치로 이행한다. 실제로는 Create New Folder도구띠단추로 metric라는 새로운 홀더를 만든다.
- metric.pro라는 파일이름을 입력하고 Save를 찰칵한다. 프로젝트의 이름이 metric로 된다.
- OK를 찰칵하여 Project Settings대화칸을 닫는다.
- File|Save를 찰칵하여 프로젝트를 보관한다.

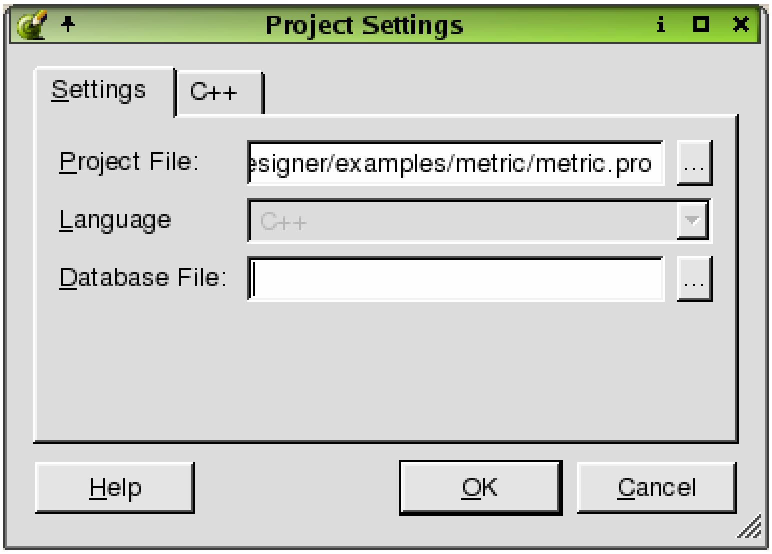


그림 1-3. Project Settings대화칸

1) 대화칸의 작성

- File|New를 찰칵하여 **New File**대화칸을 연다.
- Dialog를 선택하고 OK를 찰칵한다.
- 새 폼의 모서리를 끌기하여 좀 더 작게 만든다.
- Property Editor에서 폼의 이름을 ConversionForm으로, 제목을 Metric Conversion으로 변경한다.
- File|Save를 찰칵하여 기정폼이름을 받아들이고 Save를 찰칵하여 보관한다.

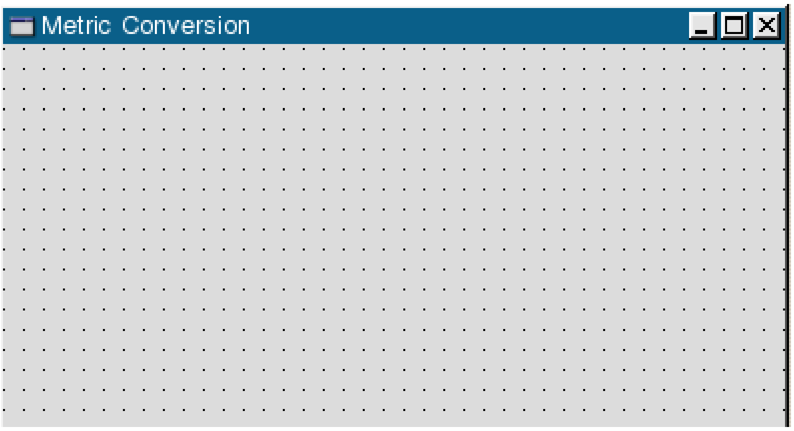


그림 1-4. 새로운 폼대화칸

2) 대화칸에 창문부품들의 추가

- 본문표식자의 추가

• Qt Designer의 기본창문의 왼쪽에 있는 Toolbox도구띠에 렬거된 Common Widgets단추를 찰칵한다.

• TextLabel단추를 련속 두번 찰칵하고 대화칸의 왼쪽웃모서리를 찰칵한다.

• 우에서 아래로 내려오면서 5개의 표식자를 배치할 때까지 맨 꼭대기의 본문표식자아래에 4번 더 찰칵한다. 표식자들의 배치는 배치도구가 자동적으로 조종하므로 여기서는 정확히 배치하지 않아도 된다.

• Pointer도구띠단추를 찰칵하여 본문표식자의 선택을 해제한다.

이제는 응용프로그람에 적합하게 본문표식자들의 속성을 변경한다.

• 제일 우에 있는 본문표식자우를 찰칵한다. Property Editor창문으로 가서 표식자의 text속성을 Enter &Number:로 변경한다. &는 다음 문자를 Alt지름건으로 만든다.

• 둘째 표식자를 선택하고 text속성을 Convert &From:로 변경한다.

• 셋째 표식자를 선택하고 text속성을 Convert &To:로 변경한다.

• 넷째 표식자를 선택하고 text속성을 Result:로 변경한다.

• 다섯째 표식자를 선택하고 text속성을 &Decimals:로 변경한다.

• File|Save를 찰칵한다.

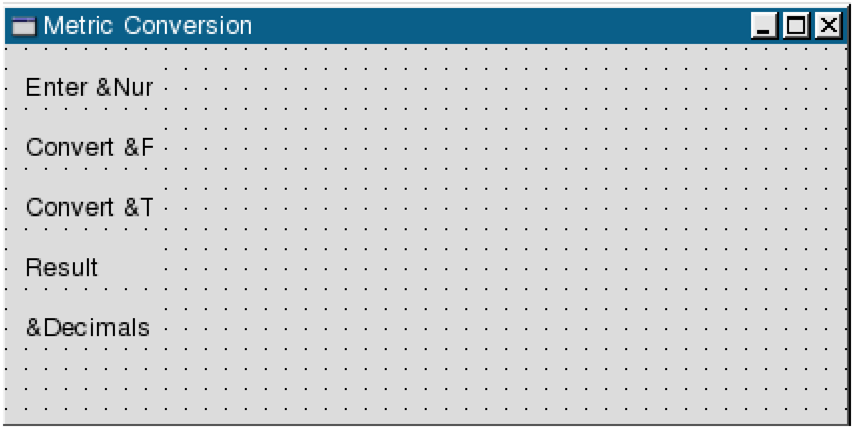


그림 1-5. 대화칸에 표식자의 추가

- 행편집칸과 복합칸, 스핀칸의 추가

• Qt Designer의 기본창문의 왼쪽에 있는 Toolbox도구띠에 렬거된 Common Widgets단추를 찰칵한다.

• LineEdit단추를 찰칵한 다음 Enter Number표식자의 오른쪽을 찰칵한다. 그 name속성을 numberLineEdit로, hAlign속성을 AlignRight로, vAlign을 AlignTop로 변경한다. font속성을 그것을 전개하여 bold 부분의 값을 조절할수 있게 강조가변(bold variant)으로 변경한다. 끝으로 wordwrap를 False로 변경한다. 창문부품을 찰칵할 때마다 그 속성이 Property Editor에 나타난다.

• ComboBox단추를 련속 두번 찰칵하고 Convert From표식자의 오른쪽을 찰칵한다. Convert To표식자의 오른쪽을 찰칵한다. Pointer도구띠단추를 찰칵하여 ComboBox선택을 해제한다. 첫째 복합칸의 name속성을 fromComboBox로 변경한다. 둘째 복합칸의 name속성을 toComboBox로 변경한다.

• LineEdit단추를 누른 다음 Result표식자의 오른쪽을 찰칵한다. 그 name속성을 resultLineEdit로 변경한다. paletteBackgroundColor속성을 누런색배경으로 하고 font속성을 강조체변경가능으로 수정하여 결과가 두드러지게 한다. hAlign을 AlignRight, vAlign을 AlignVCenter, wordwrap를 False로 한다. readOnly속성을 True로 변경한다.

• SpinBox단추를 찰칵한 다음 Decimals표식자의 오른쪽을 찰칵한다. resultLineEdit의 오른쪽아래에 SpinBox를 배치한다. 스핀칸의 name속성을 decimalsSpinBox로 변경하고 그 max value속성을 6, value를 3으로 변경한다.

• File|Save를 찰칵한다.

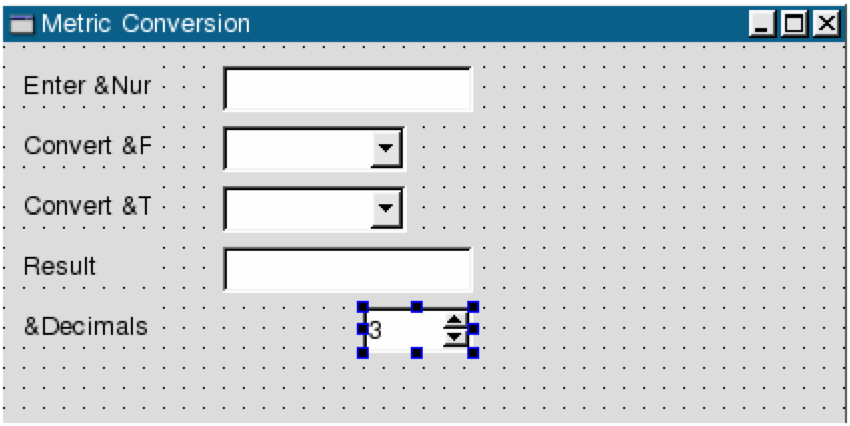


그림 1-6. 대화칸에 행편집, 복합칸, 스핀칸의 추가

이제는 매개 본문표식자를 대응하는 창문부품과 련결해야 한다. 짝패(buddy)를 작성하여 이것을 수행한다.

초점을 받아들이지 않는 창문부품 례하면 QLabel은 짝패창문부품의 이름에 첫 창문부품의 buddy속성을 설정함으로써 초점을 그 짝패, 례를 들면 QLineEdit에 넘기는 지름건을 가질수 있다.

- Set Buddy도구띠단추를 찰칵하거나 F12건을 누른다. Enter Number표식자를 누르고 선을 numberLineEdit에로 끌어다놓는다.
- Set Buddy도구띠단추를 찰칵한다. Convert From표식자를 누르고 fromComboBox에로 선을 끌어다놓는다.
- Set Buddy도구띠단추를 찰칵한다. Convert To표식자를 누르고 선을 toComboBox에로 끌어다놓는다.
- Set Buddy도구띠단추를 찰칵한다. Decimals표식자를 누르고 선을 decimalsSpinBox에로 끌고가서 놓는다.

- 누름단추의 추가

- Toolbox안에서 Common Widgets단추를 찰칵한다.
- Pushbutton단추를 두번 련속 찰칵하고 대화칸의 왼쪽아래모서리의 Decimals표식자아래를 찰칵한다.
- 새 누름단추의 오른쪽을 두번 더 찰칵하여 대화칸바닥의 한 행에 누름단추들이 수평으로 놓이게 한다.
- Pointer도구띠단추를 찰칵하여 누름단추를 해제한다.

Property Editor창문에서 각 누름단추에 대하여 몇가지 속성을 변경한다.

- 제일 왼쪽에 있는 누름단추를 찰칵하고 이름을 clearPushButton로, 본문을 &Clear로 변경한다.
- 중간의 누름단추를 찰칵하고 이름을 calculatePushButton로, 본문을

Calculate로, 기정속성을 True로 변경한다.

• 제일 오른쪽 누름단추를 찰칵하고 이름을 quitPushButton로, 본문을 &Quit로 변경한다.

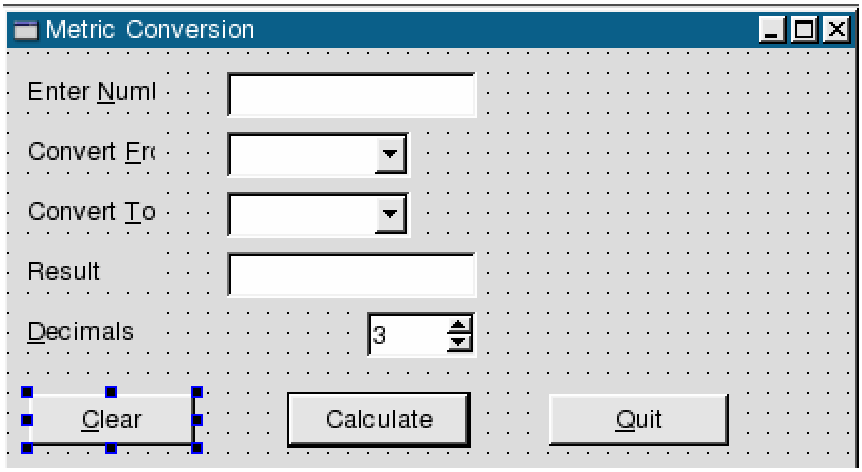


그림 1-7. 대화칸에 누름단추의 추가

- 수축자의 추가

대화칸에서 수축자(spacer)들을 추가하여 부품들사이의 여유공간을 조절함으로써 그것들을 보기좋게 배치할수 있다.

• Toolbox에서 Common Widgets단추를 찰칵한다.

• Spacer단추를 찰칵하고 Decimal표식자의 오른쪽을 누르고 스핀칸쪽으로 오른쪽으로 끌고가서 놓는다. 수평수축자가 표시된다.

• Spacer단추를 다시 찰칵하고 Calculate누름단추의 오른쪽을 누르고 Quit누름단추쪽으로 오른쪽으로 끌고가서 놓는다.

• Spacer단추를 찰칵하고 스핀칸 바로 아래를 누르고 단추들의 방향으로 수직으로 끌고가서 마우스를 놓는다. 수직수축자가 나타난다.

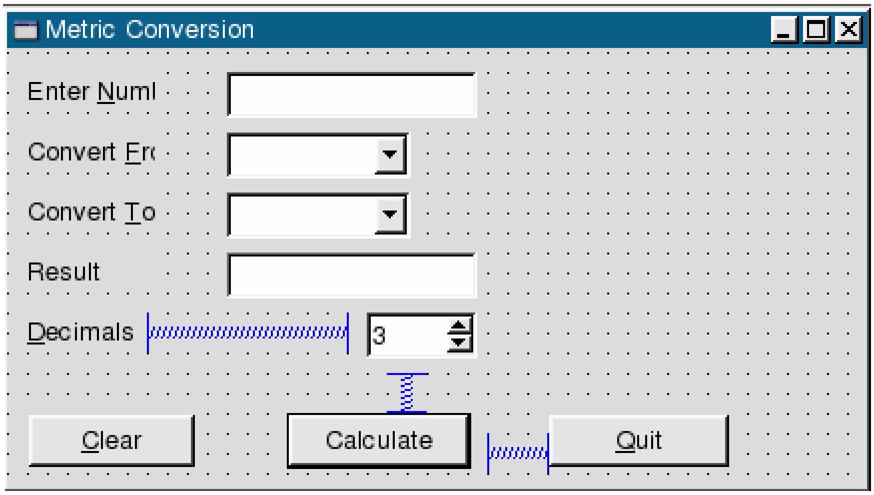


그림 1-8. 대화칸에 수축자의 추가

File|Save를 눌러서 우에서 설정한 내용들을 보존한다.

- 창문부품의 편집

이제는 창문부품들을 편집하여 변환에 필요한 값들을 포함하게 한다.

fromComboBox:

- fromComboBox를 오른쪽 단추찰칵하여 열리는 문맥차림표로부터 Edit를 찰칵한다.
- New Item을 찰칵한 다음 New Item본문을 삭제하고 Kilometers로 바꾼다.
- New Item을 찰칵하고 본문을 Meters로 변경한다.
- New Item을 찰칵하고 본문을 Centimeters로 변경한다.
- New Item을 찰칵하고 본문을 Millimeters로 변경한다.
- OK를 찰칵하여 Edit Listbox대화칸을 닫는다.

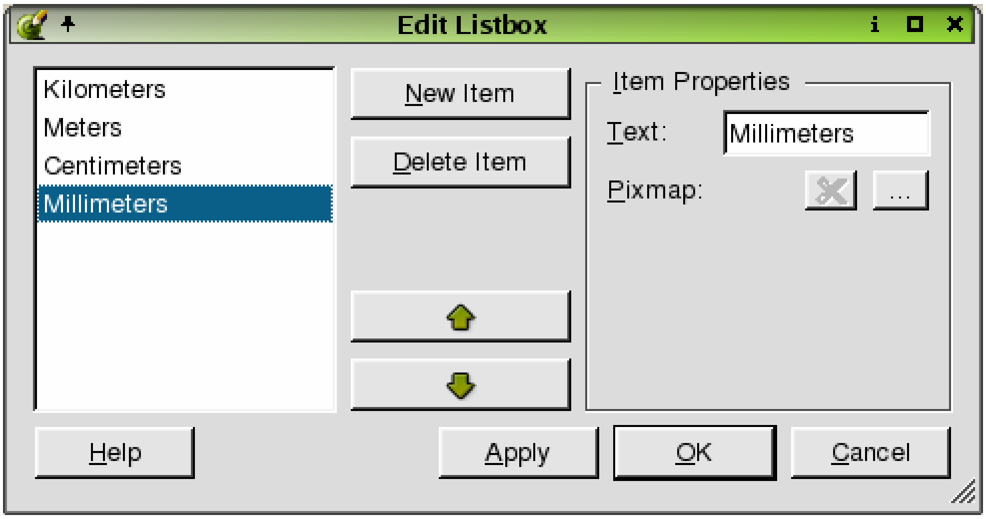


그림 1-9. 창문부품의 편집

toComboBox:

• toComboBox를 오른쪽 단추로 찰칵하면 열리는 문맥차림표에서 Edit를 찰칵한다.

• New Item본문을 삭제하고 Miles라고 입력한다.

• New Item을 찰칵하고 본문을 Yards로 변경한다.

• New Item을 찰칵하고 본문을 Feet로 변경한다.

• New Item을 찰칵하고 본문을 Inches로 변경한다.

• OK를 찰칵하여 Edit Listbox대화칸을 닫는다.

3) 대화칸의 배치

우선 본문표식자들과 그에 대응하는 창문부품들을 배치하고 마지막에 누름단추를 배치한다.

• decimalsSpinBox를 선택하고 그 다음에 있는 수축자우에서 Shift를 누르면서 마우스단추를 찰칵한다.

• Lay Out Horizontally (Ctrl+H)도구띠단추를 찰칵한다.

• 폼을 찰칵하여 이미 선택상태를 해제한다.

• Decimals본문표식자의 왼쪽에 있는 폼부분을 찰칵하고 decimals수축자는 포함되지만 다른 수축자들과 누름단추들은 피하면서 모든 표식자과 창문부품이 닿도록 선택창을 끌고간다. 마우스를 놓고 요구되는 창문부품들이 선택되였다는것을 알린다.

• Lay Out in a Grid (Ctrl+G)도구띠단추를 찰칵한다.

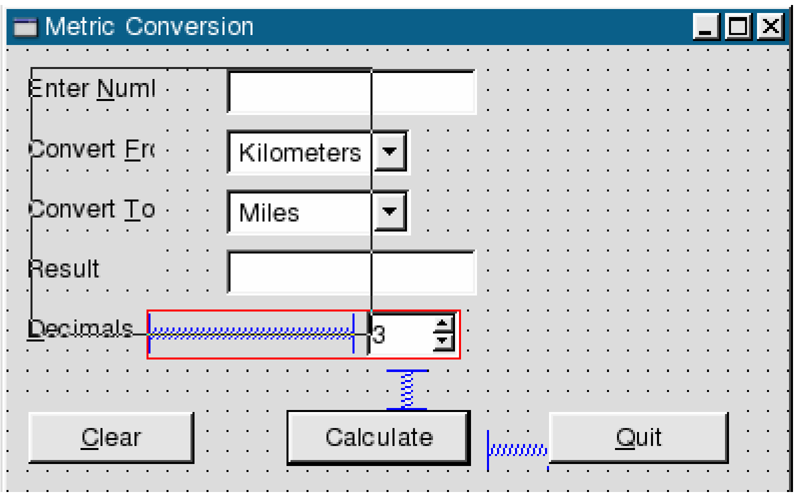


그림 1-10. 살창배치

• Object Explorer창문(Objects타브)에서 Clear누름단추를 찰칵하고 그다음 Shift를 누르면서 Calculate누름단추, Quit누름단추, 누름단추들사이의 수축자를 마우스로 선택한다.

• Lay Out Horizontally도구띠단추를 찰칵한다.

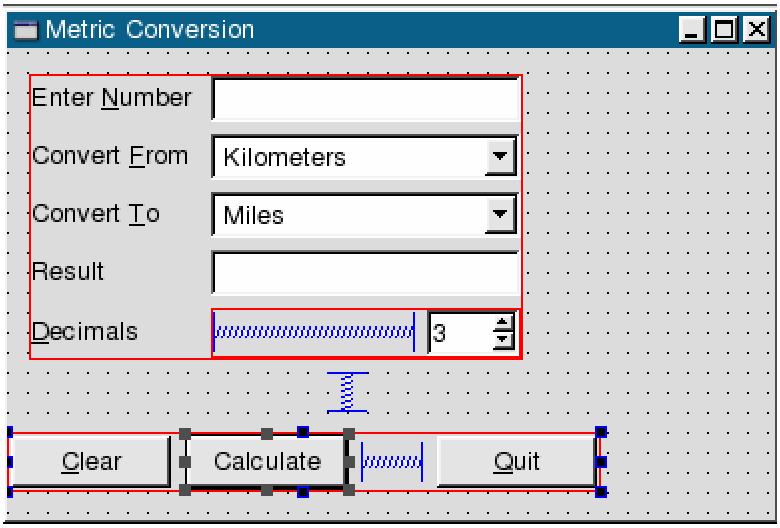


그림 1-11. 누름단추들의 수평배치

• 폼을 눌러서 모든 창문부품선택과 배치를 해제하고 Lay Out Vertically(Ctrl+L) 도구띠단추를 찰칵한다.

• 끝으로 폼을 선택하고 Adjust Size(Ctrl+J)도구띠단추를 찰칵한다.

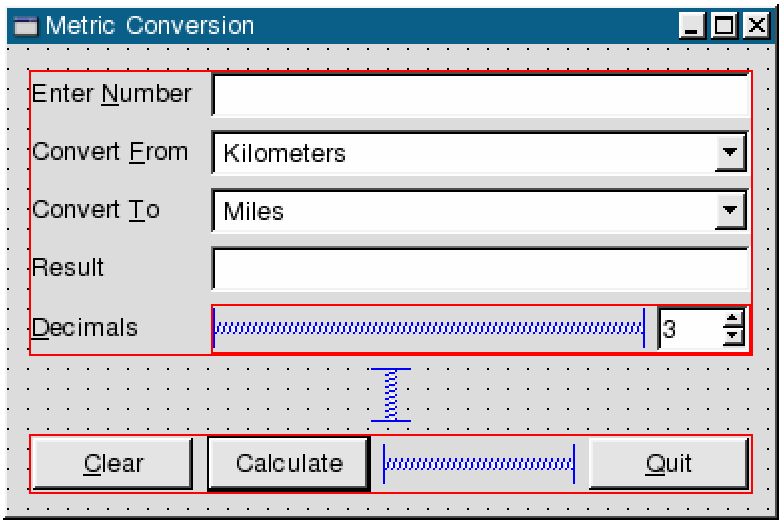


그림 1-12. 수직배치

File|Save를 눌러서 대화칸을 보관한다.

- 타브순서

대화칸의 타브순서가 정확히 설정되였는가 검사한다.

• Tab Order도구띠단추를 찰칵한다. 수자가 표시된 푸른색의 원이 초점을 받아들이는 매개 창문부품의 옆에 표시된다.

• 초점을 받아들이는 순서로 매개 창문부품들을 찰칵한다.

• Esc건을 눌러서 타브순서방식에서 탈퇴한다.

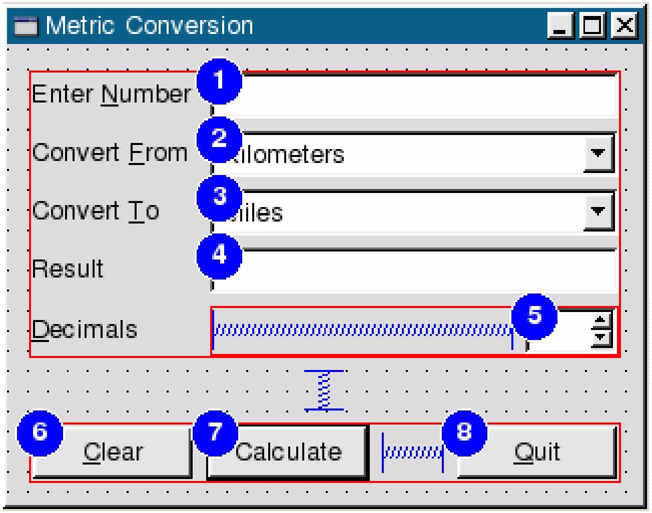


그림 1-13. 타브순서

4) 대화칸의 미리보기

대화칸을 미리 보려면 Ctrl+T를 누르거나 도구띠에서 Preview|Preview Form을 찰칵한다. 대화칸의 모서리를 끌기하여 크기를 조절한다. 창문부품들은 항상 대화칸의 크기에서 문제가 없도록 비례에 맞게 배치된다. Tab건을 눌러서 창문부품들의 타브순서를 검사한다.

5) 창문부품의 련결

3개의 단추 즉 Clear단추, Calculate단추, Quit단추를 련결해야 한다. 또한 일부 다른 창문부품들도 련결해야 한다. 보통 View and Edit Connections대화칸을 사용하여 모든 련결을 수행한다.

이제 clearButton을 련결한다.

- Edit|Connections을 찰칵하여 View and Edit Connections대화칸을 펼친다.
- New를 찰칵하고 새로운 련결을 입력한다.
- 첫 련결에 대하여 Sender로서 clearPushButton, Signal로서 clicked(), Receiver로서 numberLineEdit, Slot로서 clear()를 선택한다.
- 다시 New를 찰칵한다.
- Sender로서 clearPushButton, Signal로서 clicked(), Receiver로서 resultLineEdit, Slot로서 clear()를 선택한다.
- 다시 New를 찰칵한다.
- Sender로서 clearPushButton, Signal로서 clicked(), Receiver로서 numberLineEdit, Slot로서 setfocus()를 선택한다.

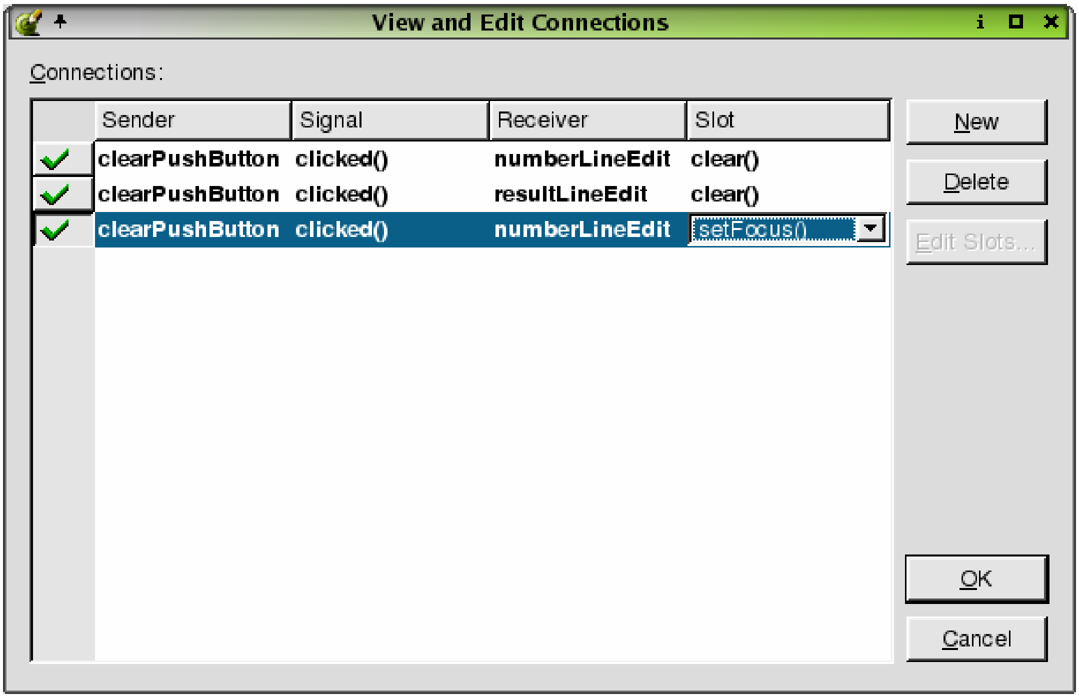


그림 1-14. clearButton의 련결

또한 중지단추를 폼과 련결한다.

• New를 찰칵한다.

• Sender로서 quitPushButton, Signal로서 clicked(), Receiver로서 ConversionForm, Slot로서 close()를 선택한다.

Calculate단추와 다른 창문부품들을 련결하려고 하지만 사용하려는 처리부는 복합칸에 렬거되지 않는다. 새로운 처리부를 만들고 목록에서 그것을 선택하여 련결을 완료한다.

• New를 찰칵한다.

• Sender로서 calculatePushButton, Signal로서 clicked(), Receiver로서 ConversionForm을 선택한다.

• Edit Slots를 찰칵하여 Edit Functions대화칸을 연다.

• New Function를 찰칵한다.

• Function name에 새 함수이름 convert()를 입력한다.

• OK를 찰칵한다.

Slot복합칸에서 convert()를 선택하여 련결을 완성한다.

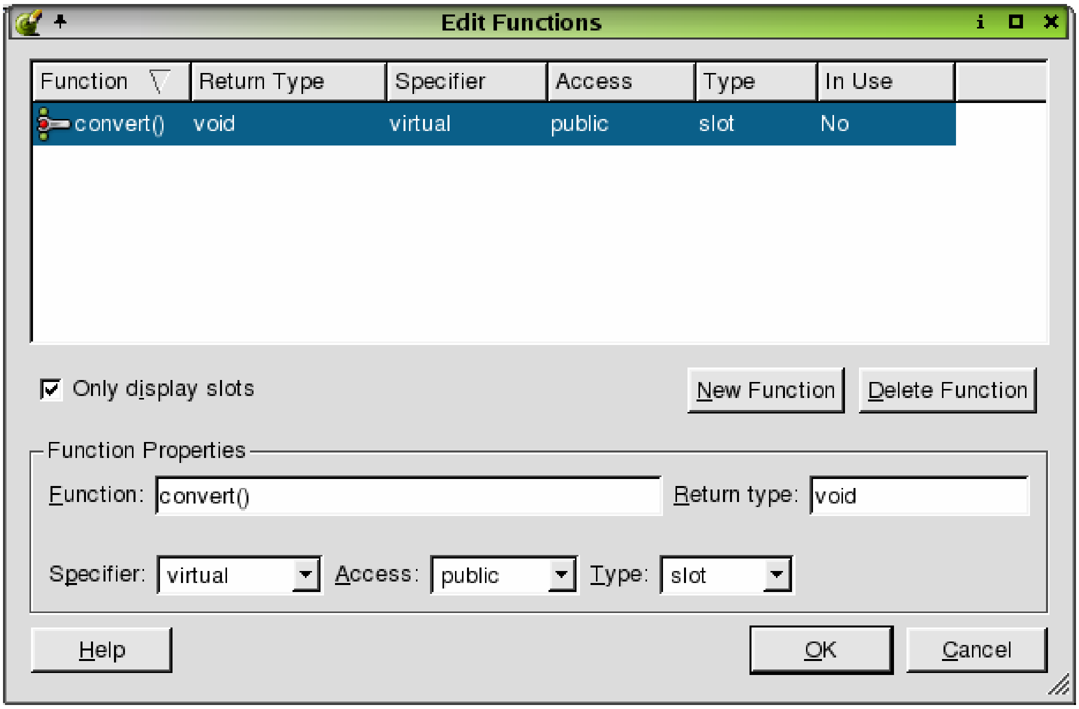


그림 1-15. 함수의 편집

이제는 마지막으로 여러개의 창문부품들을 련결한다.

- New를 찰칵한다.
- Sender로서 decimalsSpinBox, Signal로서 valueChanged(int), Receiver로서 ConversionForm, Slot로서 convert()를 선택한다.
- New를 찰칵한다.
- Sender로서 fromComboBox, Signal로서 activated(int), Receiver로서 ConversionForm, Slot로서 convert()를 선택한다.
- New를 찰칵한다.
- Sender로서 toComboBox, Signal로서 activated(int), Receiver로서 ConversionForm, Slot로서 convert()를 선택한다.
- New를 찰칵한다.
- Sender로서 calculatePushButton, Signal로서 clicked(int), Receiver로서 numberLineEdit, Slot로서 setFocus()를 선택한다.

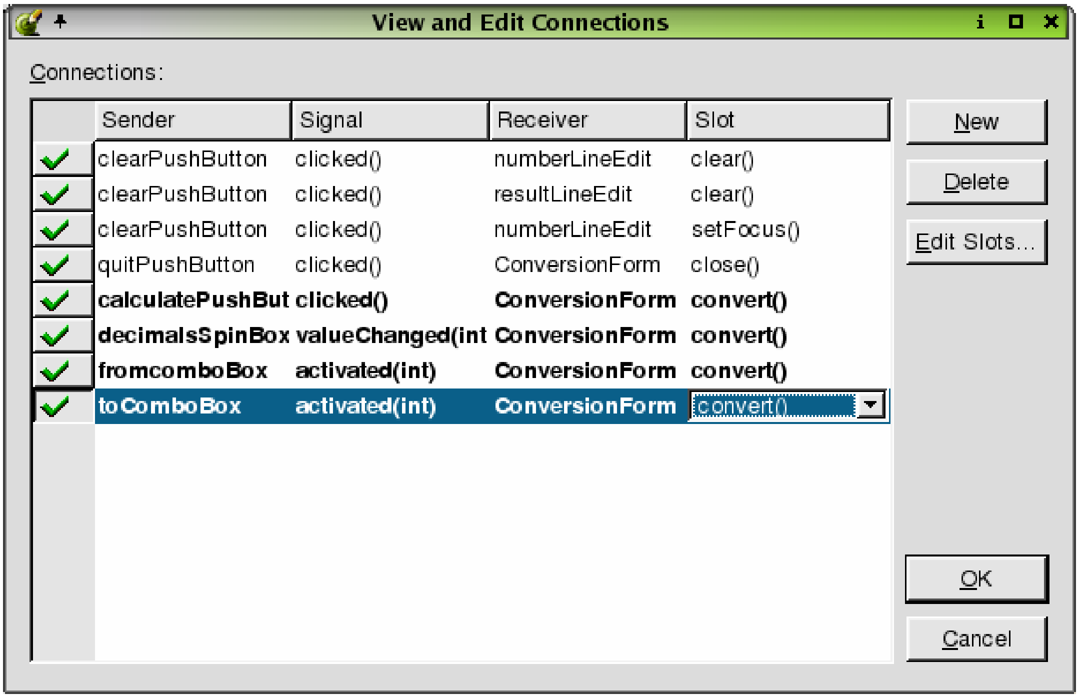


그림 1-16. 창문부품들과 ConversionForm의 convert와의 련결

- OK단추를 찰칵하여 View and Edit Connections대화칸을 닫는다.
- Save를 눌러서 프로젝트를 보관한다.

3. 대화칸코드작성

Project Overview창문에서 conversionform.ui.h를 눌러서 코드편집기를 연다. convert()함수와 init()함수를 실현한다.

```
void ConversionForm::convert()
{
  enum MetricUnits {
    Kilometers,
    Meters,
    Centimeters,
    Millimeters
  };
  enum OldUnits {
    Miles,
    Yards,
    Feet,
    Inches
```

```
};

// 입력을 얻는다
double input = numberLineEdit->text().toDouble();
double scaledInput = input;

// 내부적으로 입력을 mm로 변환한다
switch ( fromComboBox->currentItem() ) {
case Kilometers:
   scaledInput *= 1000000;
   break;
case Meters:
   scaledInput *= 1000;
   break;
case Centimeters:
   scaledInput *= 10;
   break;
}

// in으로 변환한다
double result = scaledInput * 0.0393701;

switch ( toComboBox->currentItem() ) {
case Miles:
   result /= 63360;
   break;
case Yards:
   result /= 36;
   break;
case Feet:
   result /= 12;
   break;
}
```

```
    // 결과를 설정한다
    int decimals = decimalsSpinBox->value();
    resultLineEdit->setText( QString::number( result, 'f', decimals ) );
    numberLineEdit->setText( QString::number( input, 'f', decimals ) );
}
```

우선 입력과 출력단위를 위한 렬거형을 정의한다. 그다음 numberLineEdit로부터 입력을 얻는다. 입력을 가장 정확한 메터계단위인 미리메터로 변환한다. 그다음 가장 정확한 출력단위인 인치로 변환한다. 그다음 선택된 출력단위로 그것을 비례화한다. 끝으로 결과를 resultLineEdit에 넣는다.

다음으로 대화칸이 창조될 때 호출되는 init()함수를 실현한다.

```
void ConversionForm::init()
{
    numberLineEdit->setValidator( new QDoubleValidator( numberLineEdit) );
    numberLineEdit->setText( "10" );
    convert();
    numberLineEdit->selectAll();
}
```

이 함수에서는 numberLineEdit에 대하여 유효성확인기를 설정하여 사용자가 수만 입력할수 있게 한다. 또한 이것을 수행하기 위하여 폼의 짝대기, 바로 init()우에 #include <qvalidator.h>를 추가해야 한다. 끝으로 초기입력을 설정한다.

응용프로그람을 실행할 준비가 거의 되였다. 응용프로그람을 콤파일하기전에 main.cpp파일이 요구된다.

File|New를 눌러서 New File대화칸을 연다.

C++Main File(main.cpp)을 선택하고 OK를 찰칵한다.

Configure Main-File대화칸와 련관된 기정선택을 받아들인다.

Save를 눌러서 프로젝트를 보관한다.

- 응용프로그람의 콤파일과 실행

기동하거나 콘솔로 절환하고 프로젝트를 보관한 등록부로 넘어간다.

qmake -o Makefile metric.pro라고 입력하여 프로젝트파일(metric.pro)에 대하여 qmake를 실행한다.

make를 실행한다.(혹은 체계에 따라서 nmake를 실행한다.)

프로젝트를 콤파일한 다음 응용프로그람을 실행한다.

이 응용프로그람은 Qt Designer에 대한 간단한 소개와 폼에 창문부품의 추가, 창문부품속성의 설정, 신호-처리부련결만들기, 전용코드의 배치와 추가에 대한 소개를 주었다.

제2절. 기본창문응용프로그람의 작성

2절과 3절에서는 colortool이라고 부르는 작으면서도 완전한 Qt응용프로그람을 작성한다. colortool응용프로그람은 이름을 색과 결합하는데 사용된다. 이것은 사용자와의 대화칸을 편리하게 하는 전용대화칸을 가지는 표준기본창문응용프로그람을 구성한다. 이 절에서는 기본창문을 학습하고 다음 절에서는 응용프로그람을 완성하는 대화칸을 학습한다.

1. Color Tool응용프로그람

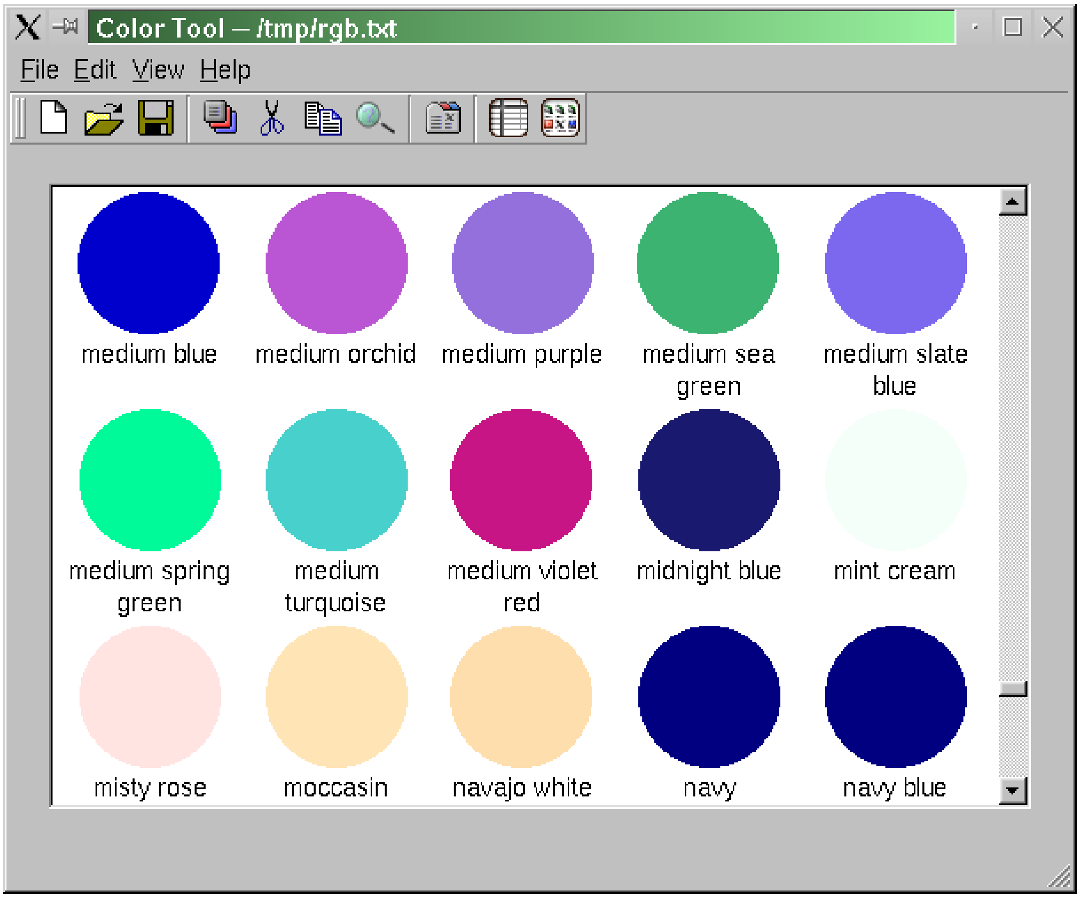


그림 1-17. 색도구의 그림기호보기

colortool응용프로그람은 다중가동환경응용프로그람으로서 사용자가 색목록을 생성하고 편집하고 보관할수 있게 한다. 매개 색은 사용자정의이름과 RGB(적, 록, 청)값을 가진다.

이 응용프로그람은 사용자에게 일련의 색과 이름을 가지는 보기를 제시한다. QWidgetStack를 사용하여 사용자가 절환할수 있는 2개의 보기를 제공한다. 표형식보기는 매개의 색을 작은 장방형과 이름, 16진값으로 보여준다. 또한 지적자의 선택을 제

공하여 색이 216표준웨브색인가 아닌가를 보여주게 한다. 그림기호보기는 매개 색을 원형색견본과 그 아래에 이름을 표시한다.

응용프로그람은 rgb.txt파일에 X체계에서 사용하는 형식으로 읽고 쓴다. 이것은 사용자가 자체의 색파일들을 생성하고 rgb.txt형식파일들을 적재, 편집, 보관할수 있게 한다.

간단한 탐색선택을 제공하여 사용자가 색을 고속으로 찾을수 있다. 이것은 수백, 수천개의 색을 표시할 때 특별히 쓸모있다. 탐색을 이행허용대화칸로 제공함으로써 사용자는 탐색과정에 기본폼과 계속 교제할수 있다. 또한 사용자가 색을 추가, 삭제하고 사용자선택을 설정할수 있게 한다. 그 편리를 도모하기 위하여 이행금지대화칸을 생성해야 한다.

끝으로 응용프로그람이 기동할 때 사용자선택을 적재하고 완료할 때 사용자선택을 보관하도록 담보해야 한다. 또한 이 선택에는 보기와 기본창문의 크기와 위치를 포함함으로써 응용프로그람은 항상 사용자가 마지막에 리용하였던 크기, 위치, 보기로 기동할수 있다.

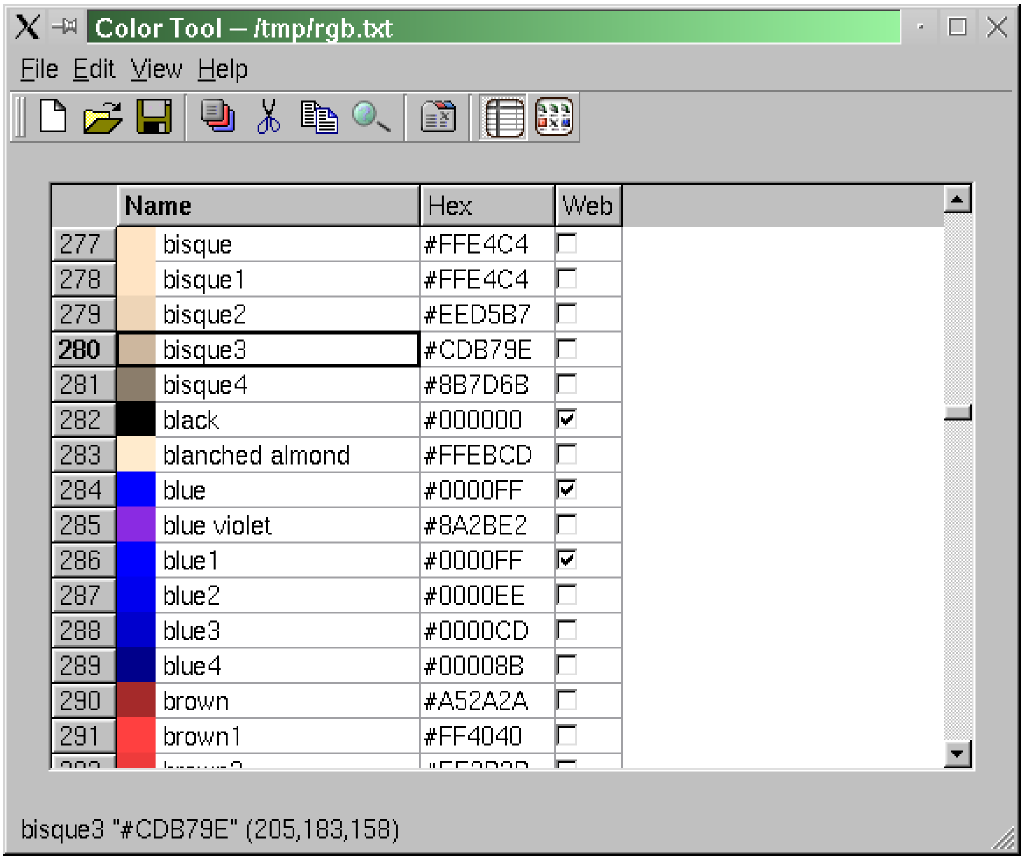


그림 1-18. 색도구의 표형식보기

2. Qt Designer의 기동과 완료

Windows하에서 Qt Designer를 기동하려면 Start단추를 찰칵하고 Programs|Qt X.x.x|Designer(X.x.x는 Qt판번호, 례하면 3.1.0)를 선택한다. Unix나 Linux조작체계를 실행하고있다면 Qt Designer그림기호를 두번 련속 누르거나 xterm에서 designer &라고 입력한다.

Qt Designer가 기동할 때 New/Open대화칸이 표시된다. Qt Designer를 다음에 열 때 이 대화칸의 표시를 요구하지 않는다면 "Don't show this dialog in the future" 검사칸을 설정한다.

이 실례에서 Cancel을 찰칵하여 대화칸을 무시한다.

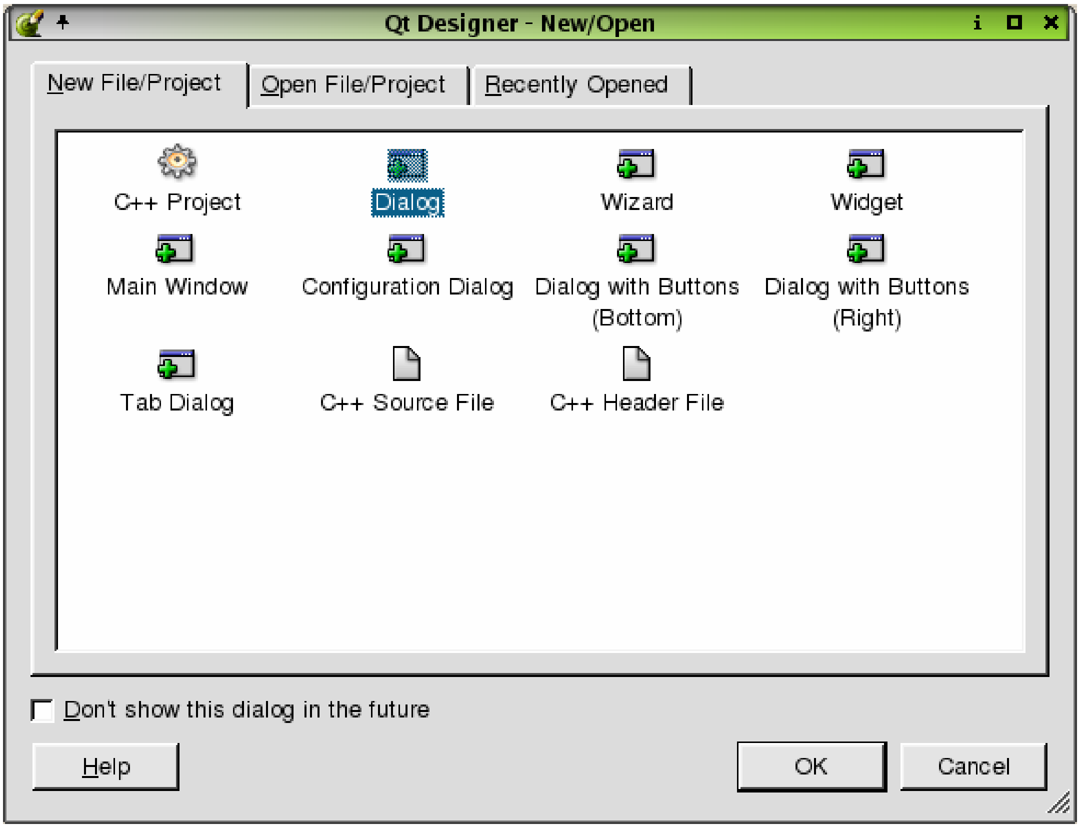


그림 1-19. New/Open대화칸

Qt Designer의 사용이 끝나면 File|Exit를 찰칵한다. 변경을 보관하지 않았으면 그것을 보관하겠는가를 묻는다. F1를 누르거나 Help차림표로부터 방조를 얻을수 있다

이 장을 정확히 리해하려면 Qt Designer를 기동하고 colortool응용프로그람을 작성해보아야 한다. 대부분의 작업은 Qt Designer의 차림표, 대화칸와 편집기의 리용을 요구한다. 작업할 때 Qt Designer의 코드편집기를 리용하여 코드를 직접 입력하여야 한다.

기정으로 Qt Designer를 기동할 때 차림표띠와 각종 도구띠를 볼수 있다. 왼쪽에 있는것이 창문부품 Toolbox이다. 도구칸의 단추들을 선택하여 특별한 도구일식을 표시한다. 오른쪽에 3개의 창문이 있다. 즉 첫째는 Project Overview창문, 둘째는 Object Explorer창문, 그리고 셋째는 Properties Editor/Signal Handlers창문이다. Project Overview창문은 프로젝트와 련관된 파일과 화상들을 표시한다. 임의의 폼(.ui파일)을 열거나 그와 련결된 코드(.ui.h파일)를 열려면 간단히 그것을 한번 찰칵한다. Object Explorer창문은 현재 폼의 창문부품과 성원들을 렬거한다. Properties Editor/Signal Handlers창문은 폼과 창문부품의 속성을 보고 변경하는데 사용된다. 실례응용프로그람을 작성할 때 Qt Designer의 창문, 대화칸, 차림표선택, 도구들의 사용법을 설명한다.

3. 프로젝트작성

colortool응용프로그람은 표준C++응용프로그람으로 만들려고 하므로 C++프로젝트를 창조하고 이 프로젝트에 파일들과 코드를 추가해야 한다.

- 프로젝트의 작성

새로운 응용프로그람을 작성할 때는 항상 프로젝트를 창조하고 개별적인 .ui파일이 아니라 프로젝트를 열것을 권고한다. 프로젝트리용은 프로젝트용으로 창조하는 모든 폼들을 파일열기대화칸을 통하여 개별적으로 적재하는것보다 마우스를 한번 찰칵하여 수행할수 있다는 우점이 있다. 프로젝트파일사용의 다른 우점은 하나의 파일에 자기 화상을 모두 보관하여 화상들이 나타나는 매개 폼에 그것들을 중복하여 보관하지 않게 하는데 있다(4절 1. 프로젝트관리).

프로젝트파일들은 .pro뒤붙이를 사용하며 qmake도구에 의하여 관련목표가동환경을 위한 makefiles을 작성하는데 사용된다.

다음과 같이 새 프로젝트를 작성한다.

File|New를 선택하여 New File대화칸을 연다.

C++ Project를 선택하여 C++프로젝트를 작성한 다음 OK를 찰칵하여 Project Settings대화칸을 연다.

Project File행편집칸의 오른쪽에 있는 생략기호단추를 찰칵하여 Save As대화칸을 연다. 이 대화칸을 사용하여 새 프로젝트를 생성하려는 곳으로 가서 Create New Folder도구띠단추를 사용하여 실제로 새 폴더(실레로 colortool)를 만든다.

파일이름을 colortool.pro라고 입력한 다음 Save를 찰칵한다. 그러면 프로젝트이름이 colortool로 된다. OK를 찰칵하여 Project Settings대화칸을 닫는다.

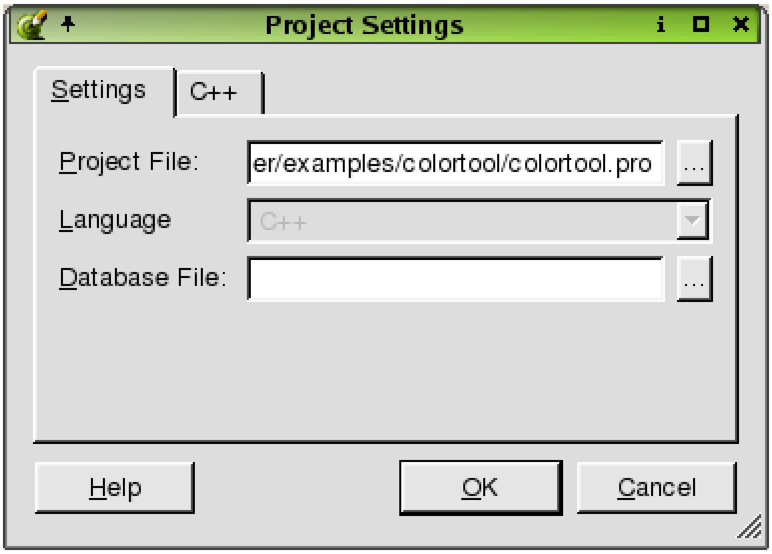


그림 1-20. Project Settings대화칸

File|Save를 눌러서 프로젝트를 보관한다.

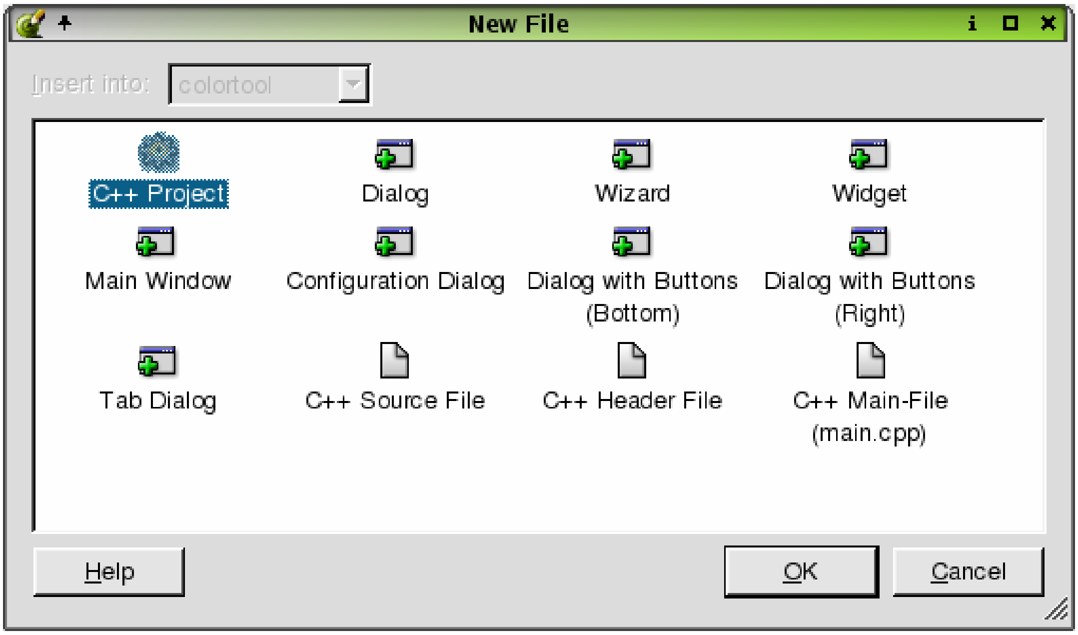


그림 1-21. New File대화칸

New File대화칸은 Qt Designer프로젝트에서 사용할수 있는 모든 파일들을 창조하는데 사용된다. 여기에는 C++원천파일, 자동생성된 main.cpp파일, 미리 정의된 형판에 기초하는 각종 폼들이 포함된다. (자체의 형판들도 만들수 있다.)

colortool응용프로그람에서는 기본창문폼으로 기동하려고 한다. 이 폼을 창조할 때 Qt Designer는 차림표와 도구띠선택을 자동적으로 생성하고 련관된 신호/처리부를 자

동생성할수 있는 위자드를 제공한다. 모든 차림표선택이나 도구띠단추에 대하여 Qt Designer는 하나의 QAction을 생성한다.

- 작용과 작용그룹

작용(action)은 사용자대면부를 통하여 사용자가 받아들이는 조작이다. 실례로 파일보존이나 본문서체를 강조체로 변경.

흔히 사용자가 각종 수단을 사용하여 작용을 처리할수 있게 하려고 한다. 실례로 파일을 보관하기 위하여 사용자가 Ctrl+S건을 누르게 하거나 Save도구띠단추를 누르게 하거나 File|Save차림표선택을 누르게 할수 있다. 작용호출수단이 다르다 할지라도 그 기초조작은 같으며 그 조작을 수행하는 코드를 반복하여 쓰려고 하지 않는다. Qt에서는 작용이 호출될 때 적당한 함수를 호출하는 QAction객체를 창조할수 있다. 작용에 지름건(레하면 Ctrl+S)을 할당할수 있다. 또한 차림표와 도구띠에 작용을 추가할수 있다.

작용이 on/off상태를 가지면 례하면 강조체가 선택 혹은 해제이면 사용자가 상태를 변경할 때(가령 도구띠단추를 찰칵하여) 그 작용과 련관된 모든것(례하면 차림표항목과 도구띠단추)의 상태는 갱신된다.

일부 작용은 라지오단추처럼 함께 조작하여야 한다. 실례로 왼쪽맞추기, 중심맞추기, 오른쪽맞추기작용들이 있다면 임의의 시각에 오직 하나만 on으로 되여야 한다. 작용그룹(QActionGroup객체)은 일련의 작용들을 하나로 묶는데 사용된다. 작용그룹의 배타적인 속성이 TRUE이면 그룹안의 단 하나의 작용이 임의의 시각에 on일수 있다. 사용자가 배타적속성이 참인 작용그룹의 한 작용의 상태를 변경하면 작용그룹안의 그 작용과 련관된 모든것, 실례로 차림표항목과 도구띠단추들이 갱신된다.

Qt Designer는 시각적으로 작용과 작용그룹을 만들고 그것들에 지름건을 할당하고 차림표항목들이나 도구띠단추들과 련결할수 있다.

4. 기본창문의 작성

Main Window Wizard를 사용하여 기본창문을 만든다. 위자드는 사용자가 작용은 물론 작용을 호출할수 있는 차림표띠와 도구띠를 만들수 있게 한다. 또한 자체의 작용, 차림표와 도구띠단추들을 만들수 있고 기본창문부품을 기본창문에 추가할수 있다.

File|New를 눌러서 New File대화칸을 열고 Main Window를 눌러서 기본창문폼을 생성한 다음 OK를 찰칵한다. 새로운 QMainWindow폼이 만들어지고 Main Window Wizard가 올리펼쳐진다.

1) Main Window위자드의 사용

처음에 Choose available menus and toolbars페지가 나타난다. 이것은 3가지의 기정작용 즉 File작용, Edit작용 그리고 Help작용을 제시한다. 각 종류에 대하여 Qt Designer가 차림표항목, 도구띠단추, 관련된 작용을 위한 신호/처리부련결을 가지도록

선택할수 있다. 항상 작용, 차림표띠, 도구띠단추를 추가 혹은 삭제하거나 후에 련결을 추가삭제할수 있다.

여기서는 File과 Edit의 기정작용을 받아들인다. 즉 차림표항목들, 도구띠단추들 그리고 생성된 관련련결들을 가진다. 사실 후에 Edit작용을 많이 변경하지만 여전히 지금은 그것들을 생성하는것이 편리하다. 여기서는 Help작용을 가지지 않으므로 Help작용의 도구띠검사칸을 비설정상태로 한다. Next를 찰칵하여 위자드의 다음 페지로 넘어간다.

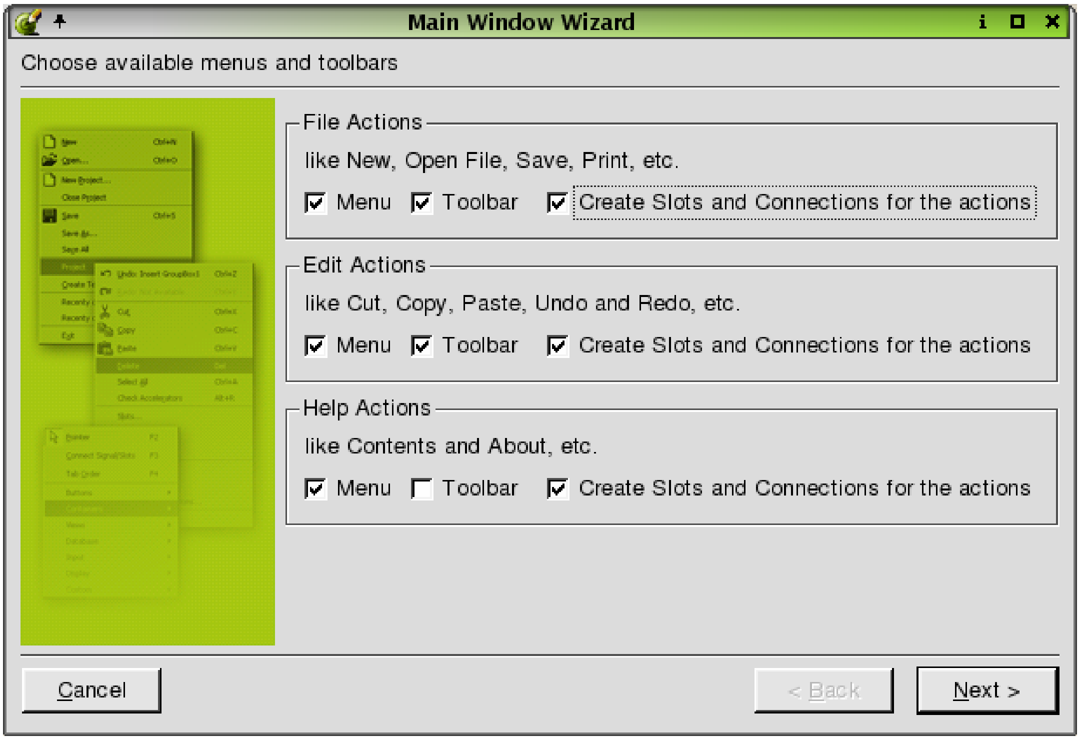


그림 1-22. Main Window Wizard-차림표와 도구띠의 선택

Setup Toolbar위자드페지는 기정작용으로부터 도구띠의 작용들을 선택하는데 사용된다. Category복합칸은 작용들의 종류를 선택하는데 사용된다. Actions목록칸은 현재 종류에서 사용할수 있는 작용들을 표시한다. Toolbar목록칸은 만들려는 도구띠단추들을 표시한다. 하늘색의 왼쪽, 오른쪽 화살단추들은 Toolbar목록칸의 안이나 밖으로 작용들을 이동하는데 사용된다. 청색의 올리, 내리화살단추들은 Toolbar목록칸안에서 작용들을 우로 혹은 아래로 옮기는데 사용된다. Actions목록칸안의 <Separator>항목은 필요할 때 Toolbar목록칸까지 이동할수 있으며 완료된 도구띠에 구분선이 보이게 한다.

Toolbar목록칸에 New와 Open, Save작용들을 복사한다. Toolbar목록칸에 <Separator>를 하나 복사한다. Category를 Edit로 바꾸고 Toolbar목록칸에 Cut와 Copy, Find작용들을 복사한다. Next를 누른 다음 Finish를 찰칵한다.

File|Save를 찰칵하고 폼을 mainform.ui로 보관한다.

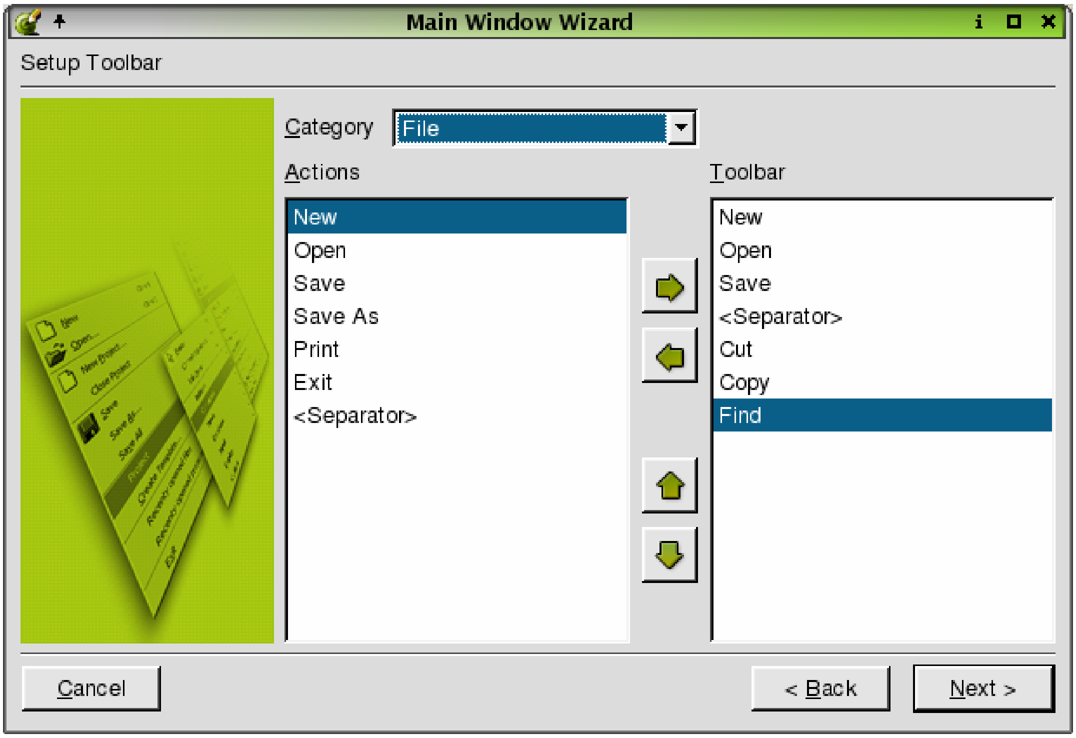


그림 1-23. Main Window Wizard-도구띠설정

폼을 미리보기(Ctrl+T)하고 File과 Edit차림표들을 사용할수 있으며 도구띠를 자기의 독립적인 창문으로 끌고가거나 그것을 창문의 좌, 우, 아래, 우로 류동할수 있다. 차림표와 도구띠는 아직 동작하지 않지만 점차 이것을 수정한다. 폼의 Close칸을 눌러서(혹은 가동환경에 고유한 조작을 하여) 미리보기방식을 완료한다.

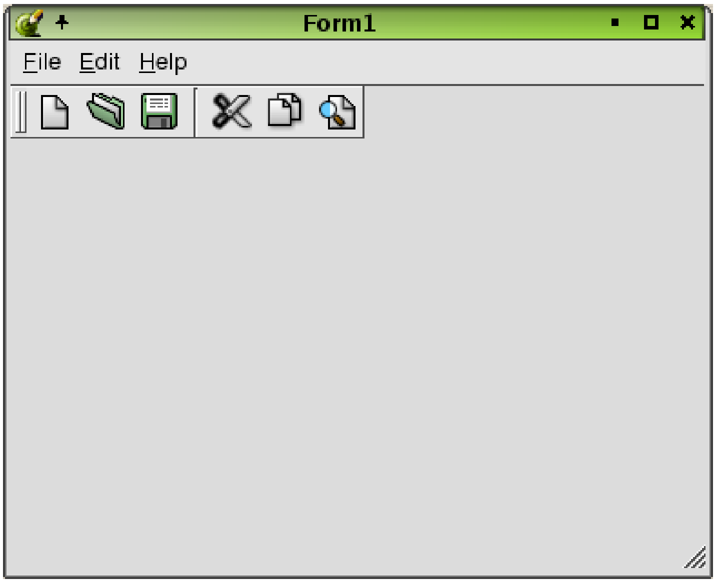


그림 1-24. 폼의 미리보기

이제는 폼을 만들었으므로 그 속성을 일부 변경하여야 한다.

- Property Editor의 사용

Property Editor는 2개 렬을 가지는데 Property렬은 속성이름을 렬거하고 Value렬은 속성값들을 렬거한다. 일부 속성이름의 왼쪽에는 네모칸안에 +가 들어있는 기호가 있는데 이것은 속성이름이 관련된 속성들의 모임에 대한 이름이라는것을 가리킨다. 폼이나 창문부품을 눌러서 Property Editor에 그 폼이나 창문부품의 속성들이 표시되게 한다.

실례로 sizePolicy속성의 +기호를 선택하면 sizePolicy아래에 4개의 속성 즉 hSizeType, vSizeType, horizontalStretch과 verticalStretch가 나타나는것을 보게 된다. 이 속성들은 다른 속성들과 같은 방법으로 편집된다.

창문부품들의 전체모임에 대하여 같은 속성을 변경하려고 한다면(실례로 그것들에게 모두 공통유표, 도구암시, 색 등을 주려고 한다면) 그 창문부품들중 하나를 선택하고 Shift와 함께 마우스를 찰칵하여 다른것들도 모두 선택한다. (혹은 Object Explorer에서 첫 창문부품의 이름을 찰칵하고 Shift와 함께 마우스를 찰칵하여 Object Explorer의 다른것들도 모두 선택한다. 이 수법은 많은 겹쌓인 창문부품들과 배치를 가지는 폼들에 특별히 쓸모있다.) 그것들이 공통적으로 가지는 속성들은 속성편집기에 표시되고 하나의 속성에 대한 변경은 선택된 모든 창문부품들을 그와 같은 속성으로 만든다.

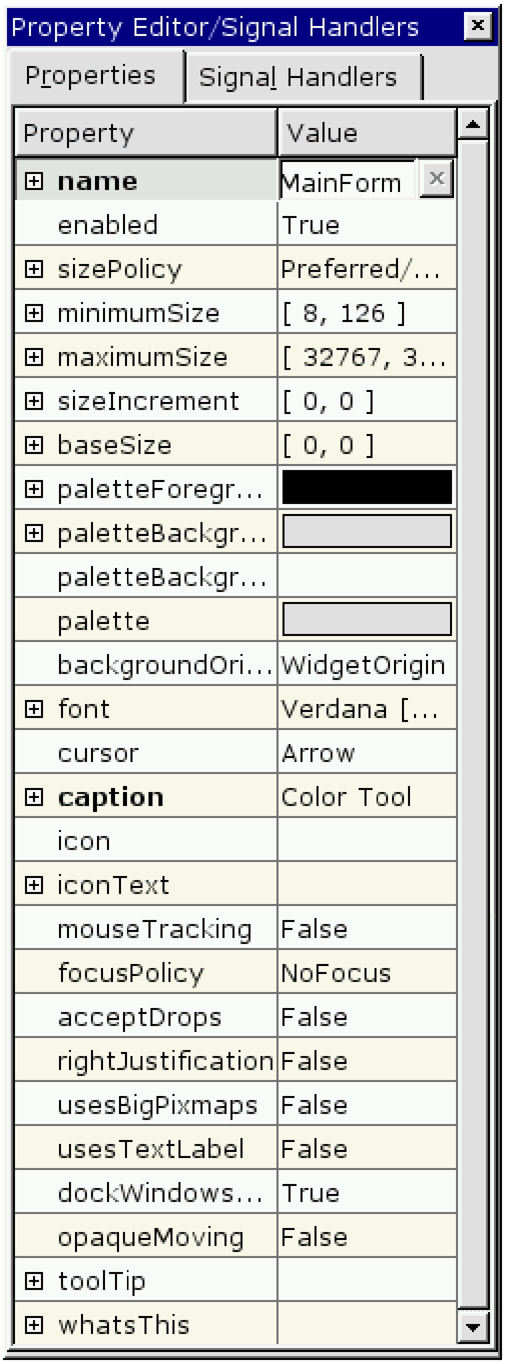


그림 1-25. Property Editor

일부 속성들은 단순값을 가진다. 실례로 name속성은 본문값을 가지며 width속성(minimumSize내에서)은 수값을 가진다. 본문값을 변경하려면 현존본문을 선택하고 새 본문을 입력한다. 수값을 변경하려면 그 값을 선택하고 새 수값을 입력하거나 스핀단추에 의하여 현존값을 요구하는 값에 이를 때까지 증가감소시킨다. 일부 속성은 고정된 값

목록을 가진다. 실례로 mouseTracking속성은 론리값이며 True 혹은 False값을 가질 수 있다. 또한 유표속성은 고정값목록을 가진다. cursor속성과 mouseTracking속성을 누르면 내리펼침복합칸에 값이 표시된다. 내리화살표를 눌러서 어떤 값이 유효한가를 볼 수 있다. 일부 속성들은 값 혹은 특수값들의 복잡한 모임을 가진다. 실례로 font속성과 iconSet속성을 들수 있다. font속성을 선택하면 생략기호단추(...)가 나타난다. 이 단추를 누르면 서체설정을 변경하는데 사용할수 있는 Select Font대화칸이 펼쳐진다. 다른 속성들도 생략단추를 가지며 속성이 가질수 있는 설정내용에 따라 각이한 대화칸들이 펼쳐진다. 실례로 text속성으로 입력할 본문이 많다면 생략단추를 찰칵하여 Multi-line Edit대화칸을 펼칠수 있다.

변경된 속성이름은 강조체로 표시된다. 속성을 변경하였는데 기정값을 되살리려고 하는 경우 그 속성의 값을 선택하고 적색 X단추를 찰칵하여 값을 되살린다. 일부 속성은 초기값 실례로 TextEdit1을 가지지만 기정값은 없다. initial값을 가지지만 기정값이 없는 속성을 되살리면(적색X를 눌러서) 그 값은 속성(례하면 이름)이 비는것이 허용되지 않으면 빈칸으로 된다.

속성편집기는 Undo와 Redo (Ctrl+Z와 Ctrl+Y, 또는 Edit차림표에서 사용가능)를 완전히 유지한다.

2) 폼의 속성과 작용의 설정

폼을 선택하여 모든 속성이 Property Editor에 나타나게 한다. 폼의 name을 MainForm로, 그 caption을 "Color Tool"로 변경한다.

이제는 기본창문위자드를 생성하였으나 응용프로그람과 련관되지 않은 작용들을 삭제해야 한다.

Object Explorer의 Members타브를 선택한다. filePrint()처리부를 오른쪽 단추로 찰칵한 다음 튀여나오기차림표로부터 Delete를 찰칵한다. 같은 방법으로 editUndo()와 editRedo(), editPaste()처리부들을 삭제한다. 후에 응용프로그람에 기능을 추가할 때 새 처리부를 생성하는 방법을 보게 된다.

또한 Action Editor창문에서 이 작용들을 삭제해야 한다. filePrintAction작용을 오른쪽 단추로 찰칵하고 튀여나오기차림표에서 Delete Action을 찰칵한다. 같은 방법으로 editUndoAction과 editRedoAction, editPasteAction작용을 삭제한다.

끝으로 폼의 차림표에서 삭제한 작용들을 구분하고있던 분리선들을 삭제한다

Action Editor창문은 류동가능하므로 그것을 마음대로 류동하지 않게 하려면 Qt Designer의 류동령역(기본창문의 상, 하, 좌, 우)중 임의의곳으로 끌고가면 된다.

폼의 File차림표를 선택한다.(지금 Qt Designer의 File차림표가 아니라 새로 생성한 폼의 File차림표를 선택하고있다.) Exit차림표선택에 2개의 분리선이 있다. (File|Print선택을 삭제하기전에 사이에 분리선이 있었다.) 분리선들중 하나를 선택하

고 Delete건을 누른다. 실수하여 우연히 차림표선택을 삭제할수 있다. 잘못 삭제하였다면 Edit|Undo하여 취소한다. 폼의 Edit차림표는 제일 웃끝에 여분의 분리선을 가지고 있다. (거기에 취소와 재시행선택이 있었다.) 같은 방법으로 이 분리선을 삭제한다. 차림표선택을 잘못 삭제하였다면 Ctrl+Z를 눌러서 취소한다.

File|Save를 눌러서 폼을 보관한다.

이제는 Preview|Preview Form (또는 Ctrl+T)을 눌러서 폼을 미리볼수 있다.

- Object Explorer

Window|Views|Object Explorer를 선택하여 Object Explorer창문을 펼친다. Object Explorer에는 2개의 타브가 있는데 Objects타브는 객체의 계층을 보여주고 Members타브는 폼에 추가한 성원들을 보여준다. Objects타브안의 창문부품이름을 눌러서 창문부품을 선택하고 Property Editor에 그 속성들을 표시한다. 많은 창문부품을 가지거나 배치를 요구하는 폼들에 특히 쓸모있는 창문부품들을 Object Explorer에서 간단히 보고 선택할수 있다. 여러 창문부품은 처음것을 누른 다음 Shift를 누르면서 다른것들을 눌러서 선택할수 있다.

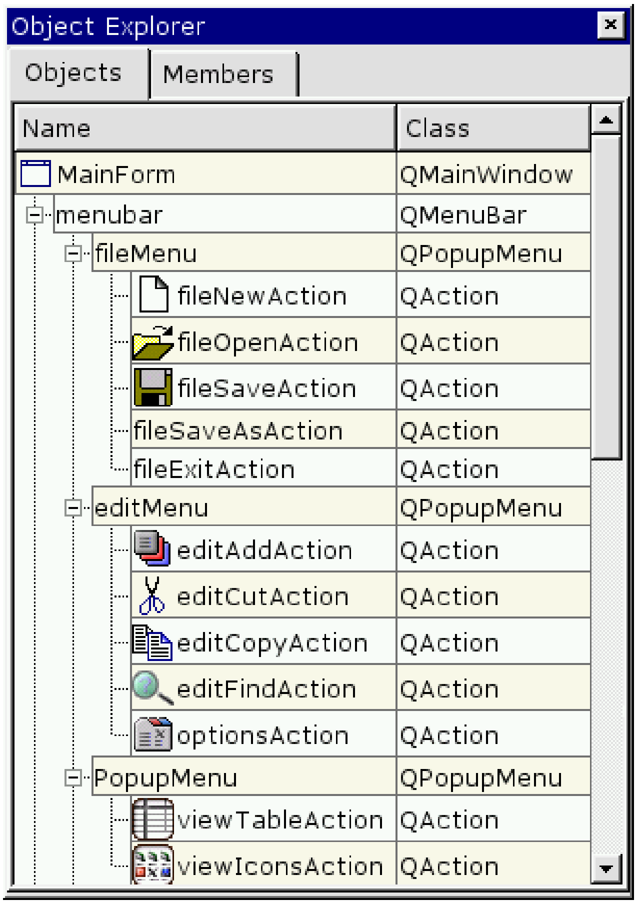


그림 1-26. Object Explorer

Qt Designer의 초판에서 폼에 코드를 제공하려고 한다면 폼의 파생클라스를 만들고 파생클라스에 코드를 추가한다. 이 판은 파생클라스작성수법을 충분히 지원하지만 폼에 코드를 직접 삽입하는 다른 수법도 제공한다. Qt Designer에서 코드쓰기는 파생클라스작성과 같이 잘되지 않는다. 실례로 폼의 구성자나 해체자에 대한 직접호출을 얻을수 없다. 구성자에 의해 실행되여야 하는 코드가 요구된다면 void init()라는 처리부를 만든다. 그것이 이미 존재하면 구성자로부터 그것을 호출한다. 마찬가지로 해체자전에 실행하려는 코드가 요구되면 void destroy()라는 처리부를 생성한다. 또한 자체의 클라스변수들을 추가하여 생성된 구성자코드에 삽입할수 있으며 필요한 앞방향선언과 머리부들을 추가할수 있다. 변수나 선언을 추가하려면 적당한 항목 례하면 Class Variables를 오른쪽 단추로 찰칵하고 New를 누른 다음 본문 례하면 QString m_filename을 입력한다. 1개이상의 항목이 존재하면 오른쪽 단추로 찰칵하여 New와 Edit, Delete선택을 가지는 차림표를 펼친다. 여러 항목 례를 들면 여러개의 머리부파일이나 여러개의 자료성원을 입력하려면 관련구간에서 오른쪽 단추를 찰칵하고 Edit를 눌러서 Edit대화칸을 호출하는것이 제일 간단하다. 코드를 편집하려면 함수이름을 눌러서 코드편집기를 펼친다. 코드편집과 처리부생성은 후에 학습한다.

폼의 파생클라스를 만들려면 자체의 .cpp파일을 만들고 거기에 필요한 구성자, 해체자, 함수들, 처리부들, 선언과 변수들을 넣는다.

3) 전용작용의 추가

응용프로그람에 고유한 작용을 사용자에게 제공하려고 할수 있다. 2개의 보기들사이를 절환하고 사용자가 색을 추가하고 자기가 좋아하는 선택을 추가하게 하려고 한다. 보기선택용의 새 차림표를 생성하고 도구띠에 분리선을 추가하여 그 방법을 준비한다.

차림표띠에서 new menu를 찰칵하고 본문에 &View라고 입력한다. &는 다음 문자에 밑선을 붙이여 Alt지름건을 만든다. (다시말하여 이 경우에 Alt+V는 View차림표를 펼친다.)



그림 1-27. 차림표띠

- 복사지름건

많은 창문부품을 포함하는 대화칸을 가지는 응용프로그람에서 지름건들이 우연히 중복될수 있다. Qt Designer는 같은 지름건들을 가지는 2개이상의 창문부품들이 강조표시되는 경우를 쉽게 검사하게 하는 Edit|Check Accelerators(Alt+R)를 제공한다.

Help차림표의 왼쪽으로 View차림표를 끌고가서 그것을 놓는다. (수직재색선은 그 위치를 가리킨다.)



그림 1-28. View차림표항목의 끌기

현존도구띠의 끝에 분리선을 넣고 분리선뒤에 View선택을 추가할 대신에 View차림표항목용의 새로운 도구띠를 생성할수 있다. 제일 오른쪽 도구띠단추(Find)를 오른쪽 단추로 찰칵하고 Insert Separator를 찰칵한다. 혹은 완전히 새로운 도구띠를 생성할수 있다.

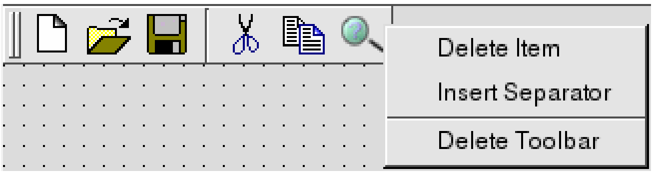


그림 1-29. 도구띠차림표에 분리선의 삽입

- 도구띠의 작성과 이식

새 도구띠는 현존 도구띠의 오른쪽을 찰칵하고 Add Toolbar를 선택하여 생성한다. 새 도구띠는 비여있고 그 도구띠핸들만 볼수 있다. (도구띠핸들은 보통 2개의 굵은 수직선이나 여러개의 자그마한 홈들을 포함하는 재색구역으로 표시된다.)

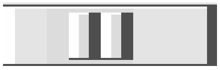


그림 1-30. 도구띠핸들

작용들은 Action Editor편집기로부터 도구띠로 그것들을 끌고가서 요구하는 위치의 도구띠에 놓는 방법으로 간단히 도구띠에 추가된다. (그 위치는 수직적색선으로 가리킨다.)

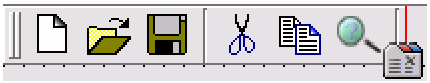


그림 1-31. 도구띠에 작용그룹을 끌고가기

작용그룹의 모든 작용들은 간단히 Action Editor로부터 작용그룹을 끌어다 도구띠우에 놓는 한번의 이행으로 도구띠에 추가된다.

보통 도구띠단추들은 오직 하나의 화상을 보여주므로 도구띠에서 사용하려는 모든 작용들은 적당한 화상에로의 iconSet속성모임을 가지고있어야 한다.

도구띠단추들과 분리선(보통 울퉁불퉁한 수직재색선들로 표시된다.)들은 임의의 시간에 도구띠의 새 위치에로 끌어다 놓을수 있다. 분리선들은 도구띠단추를 오른쪽 단추로 찰칵하고 Insert Separator를 선택하는 방법으로 삽입할수 있다. 도구띠단추들과 분리선들은 그것들을 오른쪽 단추로 찰칵하고 Delete Item을 선택하여 삭제할수 있다. 도구띠는 도구띠핸들을 오른쪽 단추로 찰칵하고 Delete Toolbar를 선택하여 삭제할수 있다.

응용프로그람을 미리보기하면 각이한 류동점(QMainWindow 혹은 파생클라스의 상, 하, 좌, 우)으로 모든 도구띠를 끌고다니거나 독립적인 도구창문들밖으로 도구띠를 끌고갈수 있다는것을 알수 있다.

- 도구띠에 창문부품의 추가

때때로 간단한 단추는 우리의 요구를 만족시키지 못한다. 실례로 사용자가 도구띠로부터 서체이름과 서체크기를 선택할수 있게 하려면 도구띠단추로 서체대화칸을 펼치는것보다 직접적인 도구를 리용하는것이 더 좋을것이다.

도구띠에 ComboBox와 SpinBox를 추가하는것은 완전히 편리하다. 실례로 ComboBox를 유효서체이름들을 렬거하는데 사용할수 있으며 SpinBox를 서체크기를 선택하는데 사용할수 있다.

도구띠에 창문부품을 넣을수 있지만 작용과 련결할수 있는 창문부품들을 도구띠에 직접 추가하지 않을것을 권고한다. 이러한 창문부품들 즉 차림표항목과 도구띠단추, 항목목록들은 작용(항목목록일 때에는 내리펼침작용)을 생성하고 작용을 창문부품과 련결하고 작용을 도구띠에 추가해야 한다.

4) Options작용의 추가

Action Editor의 첫 작용을 오른쪽 단추로 찰칵하고 New Action을 선택한다. Property Editor에는 새로운 작용의 속성들이 표시된다. 작용의 name속성을 optionsAction으로 변경한다. iconSet속성에 대한 생략단추를 찰칵하여 Choose an Image대화칸을 펼친다. Add단추를 찰칵하여 Choose Images대화칸을 연다. /tools/designer/examples/colortool/images로 넘어간다. tabwidget.png화상을 선택한다. 그것을 사용하기 위하여 Open을 선택하고 OK를 찰칵하면 Choose an Image대화칸이 열린다. text속성을 Options로, menuText속성을 &Options...로 변경한다.

Action Editor에서 Options작용을 선택하고 Edit차림표까지 끌고간다. Edit차림표는 펼쳐진다. Options작용을 그 차림표아래로 끌고가서(붉은색 수평선은 그 위치를 보여준다.) Find항목뒤의 끝에서 그것을 놓는다.

- Options작용을 추가하는 다른 수법

차림표띠에서 Edit를 찰칵하고 Find차림표뒤에 있는 new item을 찰칵하고 &Options라고 입력하여 이름을 변경하고 Enter를 입력한다. 화살건을 Options차림표항목(픽스매프마당)의 왼쪽까지 이동하고 Enter를 누르면 Choose an Image대화칸이 열린다. Add단추를 찰칵하여 Choose Images대화칸을 연다.

/tools/designer/examples/colortool/images로 이행하여 tabwidget.png화상을 선택한다. 화상을 사용하기 위하여 Open을 선택하고 OK를 찰칵하면 Choose an Image대화칸이 열린다. 그러면 픽스매프가 차림표의 Options항목다음에 나타난다.

Options작용은 다른 Edit차림표선택과 시각적으로 분리되여야 한다. 폼의 Edit차림표를 선택하고 new separator항목을 선택하고 Options항목우의 공간으로 끌고간다.

또한 이 선택을 도구띠로부터 쓸수 있게 하려고 하므로 Options작용을 선택하고 그것을 도구띠로 끌고가서 확대경(Find)도구띠단추(분리선뒤에)의 오른쪽에 놓는다. 붉은색 수평선은 끌기할 때 그 위치를 표시한다.

이 작용을 후에 련결하고 코드를 작성한다.

5) Add작용의 추가

Action Editor의 첫 작용을 오른쪽 단추로 찰칵하고 New Action을 선택한다. 작용의 name속성을 editAddAction으로, iconSet속성을 designer_widgetstack.png, text속성을 Add로, menuText속성을 &Add..., accel속성을 Ctrl+A로 변경한다. (CTRL+A를 누르면 자동적으로 건결합이 마당에 나타난다.)

Add단추를 찰칵하고 그것을 Edit차림표의 첫 항목으로 끌고간다. (그것을 편집차림표로 끌고가서 붉은색 수평선이 Cut차림표항목우에 놓일 때 놓는다.)

- Add작용의 다른 추가수법

차림표띠에서 Edit를 선택하고 Find차림표항목뒤에 있는 new item을 찰칵하고 &Add를 입력하여 이름을 변경하고 Enter를 누른다. 화살건을 Add항목왼쪽의 공간으로 이동하고 Enter를 누르면 Choose an Image대화칸이 열린다. Add단추를 찰칵하여 Choose Images...대화칸을 열고 /tools/designer/examples/colortool/images로 이행하여 tabwidget.png화상을 선택한다. 화상을 사용하기 위하여 Open을 선택하고 OK를 찰칵하면 Choose an Image대화칸이 열린다. 그러면 픽스매프가 차림표의 Options항목다음에 나타난다. 끝으로 화살건을 Add항목오른쪽의 공간으로 이동하고 Ctrl+A을 누르면 지름건결합이 Add차림표항목다음에 나타난다.

6) 정돈

Cut를 사용하여 색을 삭제할수 있지만 이름을 Delete로 변경하여 그 의미를 더 명백하게 한다. Action Editor안의 editCutAction을 선택하여 Property Editor에 그 속성들을 표시하게 한다. text속성을 Delete로, menuText속성을 &Delete로 변경한다.

그렇지만 이러한 변경은 현재 Alt+C(Cut에 사용)를 사용하지 않는다. Action Editor에서 editCopyAction작용을 선택하고 menuText속성을 &Copy로 변경한다.

- Actions작용이름을 변경하는 다른 수법

Cut작용의 이름을 Delete로 변경하기 위하여 차림표띠에서 Edit|Cut를 찰칵하고 &Delete라고 입력하고 Enter를 누른다.

Copy작용의 이름을 &Copy로 변경하려면 차림표띠에서 Edit|Copy를 찰칵하고 &Copy라고 입력하고 Enter를 누른다.

Edit|Check Accelerators(혹은 Alt+R)를 눌러서 지름건경쟁이 있는가 검사할수 있다.

7) 작용그룹의 추가

사용자가 보기들을 선택할수 있게 하려고 하지만 한번에 하나의 보기만 사용할수 있으므로 보기들사이를 절환하는데 사용하는 차림표선택과 도구띠단추들이 항상 동기된다는것을 담보해야 한다. 코드로서 이것을 달성할수 없으므로 작용그룹에 련관된 작용들을 간단히 넣는다.

Action Editor에서 오른쪽 단추로 작용을 찰칵하고 New Action Group를 선택하면 작용그룹의 속성들이 Property Editor에 표시된다. 작용그룹의 name속성을 viewActionGroup으로, 그 text속성을 View로 변경한다. 작용그룹을 배타적으로 즉 임의의 시각에 그룹의 작용들중 하나만 on으로 하려고 하지만 exclusive속성은 기정적으로 True로 설정되여있으므로 다시 설정할 필요는 없다.

그러면 보기작용들을 만들자. 이 과정은 작용그룹안에 있지 않는 작용들과 가상적으로 같다. 유일한 차이는 오른쪽 단추를 찰칵하여 문맥차림표를 펼칠 때 Action Editor의 어떤 작용이 아니라 련관된 작용그룹을 오른쪽 단추로 눌러야 한다.

viewActionGroup을 오른쪽 단추로 찰칵하고 New Action을 선택한다. 이 작용의 name속성을 viewTableAction으로 변경한다. 그 toggleAction속성을 True로, 그 on속성을 True로 설정한다. 사용자가 이 보기(on)를 사용하거나 다른 보기(off)를 사용하고있으므로 그것을 절환작용으로 하려고 한다. 이 작용이 기정보기이므로 on으로 설정한다. 그 iconSet속성을 table.png로 변경하고 text속성을 View Table로, menuText속성을 View &Table로 변경한다. accel속성을 Ctrl+T로 변경하고 toolTip속성을 View Table(Ctrl+T)로 설정한다. 사용자가 View차림표를 선택하고 View Table우로 마우스를 가져가면 도구암시가 상태띠에 나타난다. 마찬가지로 사용자가 마우스를 View Table도구띠단추우로 가져가면 도구암시본문이 상태띠와 도구띠단추곁의 황색의 일시표식자안에 나타난다.

viewActionGroup를 오른쪽 단추로 찰칵하고 New Action을 선택한다. 이 작용의 name속성을 viewIconsAction로 변경하고 그 toggleAction속성을 True, iconSet속성을 iconview.png로, text속성을 View Icons로, menuText속성을 View &Icons, accel속성을 Ctrl+I로, toolTip속성을 View Icons (Ctrl+I)로 변경한다.

8) 작용그룹의 사용

이제는 보기작용들을 만들었으므로 그것들을 사용자가 사용할수 있게 만들어야 한다.

Action Editor의 viewActionGroup작용그룹을 선택하고 View차림표로 끌고가서 이 차림표우에(View차림표아래에 붉은색 수평선이 나타날 때) 놓는다. 작용그룹을 끌기하였으므로 그의 모든 작용들(우리의 경우에 viewTableAction과 viewIconsAction)은 련관된 차림표에 추가된다. 또한 도구띠에서 보기작용들을 쓸수 있게 만든다. viewActionGroup을 다시 한번 찰칵하고 그것을 도구띠로 끌고가서 도구띠의 오른쪽에 있는 분리선의 오른쪽에 그것을 놓고 도구띠의 변두리에 놓는다. (다시 붉은색 수직선이 그 위치를 가리킨다.)

작용안의것들을 미리보는데 Ctrl+T를 사용할수 있고 정기적으로 File|Save(또는 Ctrl+S)를 찰칵해야 한다. 이제 보기도구띠단추와 차림표선택들을 누르면 도구띠단추와 차림표항목들이 모두 자동적으로 동기되는것을 볼수 있다.

5. 기본창문부품의 작성

대부분의 기본창문형 응용프로그람은 차림표띠, 도구띠, 상태띠와 중심창문부품으로 구성된다. 이미 차림표띠와 도구띠를 만들고 기본창문위자드를 통하여 QMainWindow를 만들었으므로 상태띠를 가질수 있다. 응용프로그람의 기본창문부품으로서 공통적으로 사용된 창문부품들은 QListView(나무보기를 제공)와 QTable, QTextEdit이다. 사용자들에게 같은 자료의 2가지 각이한 보기들을 제공하려고 하므로 QWidgetStack를 기본창문부품으로 사용한다. QWidgetStack는 그 자체의 시각적인 표시를 가지지 않고 각 QWidgetStack페지에 한개이상의 창문부품들을 배치하며 이때 각 페지는 하나의 폼을 가진다. 그다음 사용자에게 페지들을 절환하기 위한 기구를 제공한다. (이것은 QTabWidget를 사용하는것과 원칙적으로 비슷하다.) 우리는 사용자들에게 2개의 보기 즉 색과 그 이름들을 렬거하는 표형식보기와 색견본을 보여주는 그림기호에 기초한 보기를 제공하려고 한다. 실례에서는 각 QWidgetStack페지에 하나의 창문부품만 배치하지만 이것은 응용프로그람의 설계를 완전히 반영한다. 각 페지에 여러개의 창문부품을 배치할수 있다.

Toolbox의 Containers단추를 찰칵하고 WidgetStack를 선택한다. 폼의 거의 중간을 눌러서 창문부품탄창을 배치한다. 창문부품탄창의 name속성을 colorWidgetStack로 변경한다.

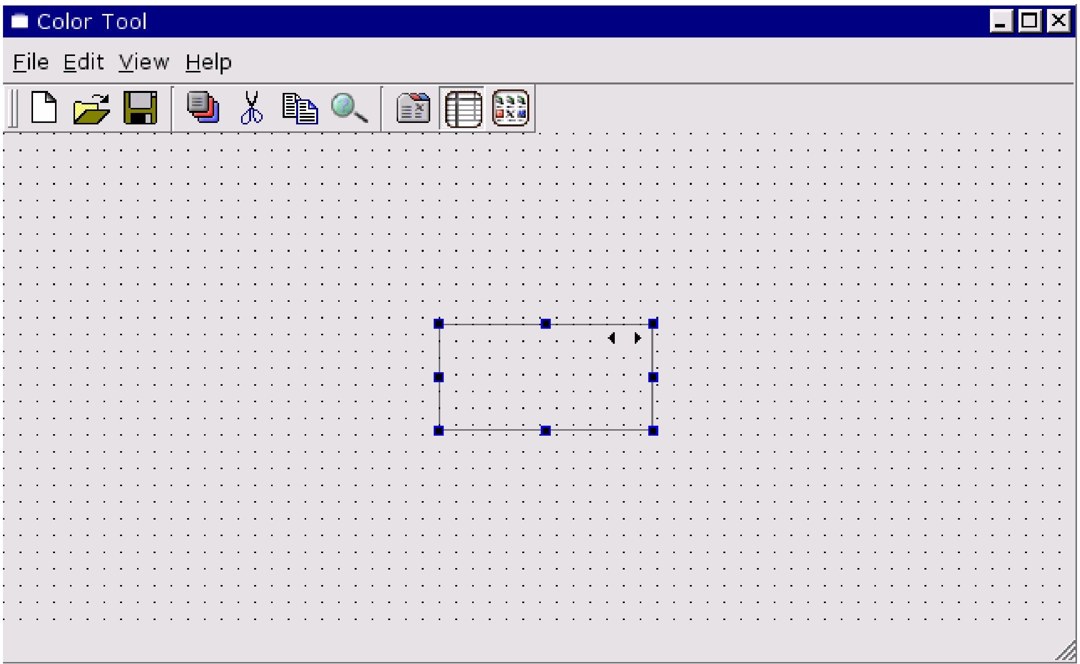


그림 1-32 .창문부품탄창의 배치

Qt Designer를 사용하여 폼에 창문부품들을 배치할 때 대체로 오른쪽에 배치해야 한다. 그리고 배치된 창문부품들의 크기에 대하여 걱정할 필요는 없다. 실례로 표식자를 하나 배치하고 그 본문을 크기에 맞지 않게 변경하여도 문제는 생기지 않는다. 정확한 위치와 크기에 대하여 걱정하지 않아도 되는 원인은 Qt Designer가 Qt의 배치클라스들을 사용하여 폼들을 자동적으로 배치하기때문이다. 우리는 창문부품들의 모임을 선택하고 Qt Designer에게 그것들이 수직 혹은 수평으로, 나란히 혹은 살창형식으로 배치해야 한다는것을 알려주어야 하며 Qt Designer는 그것들을 배치하고 크기를 적당히 조절한다.

이 절에서는 Qt Designer의 배치편의프로그람을 최소로 사용한다. 3절에서 배치의 사용법과 정보를 더 제공하고 여러개의 대화칸을 만든다.

홈자체를 선택하고 Lay Out Vertically도구띠단추를 찰칵한다. 그러면 창문부품탄창은 전체폼안에 꽉 찬다. 이제는 창문부품탄창의 페지들에 창문부품들을 이식할 준비가 되였다.

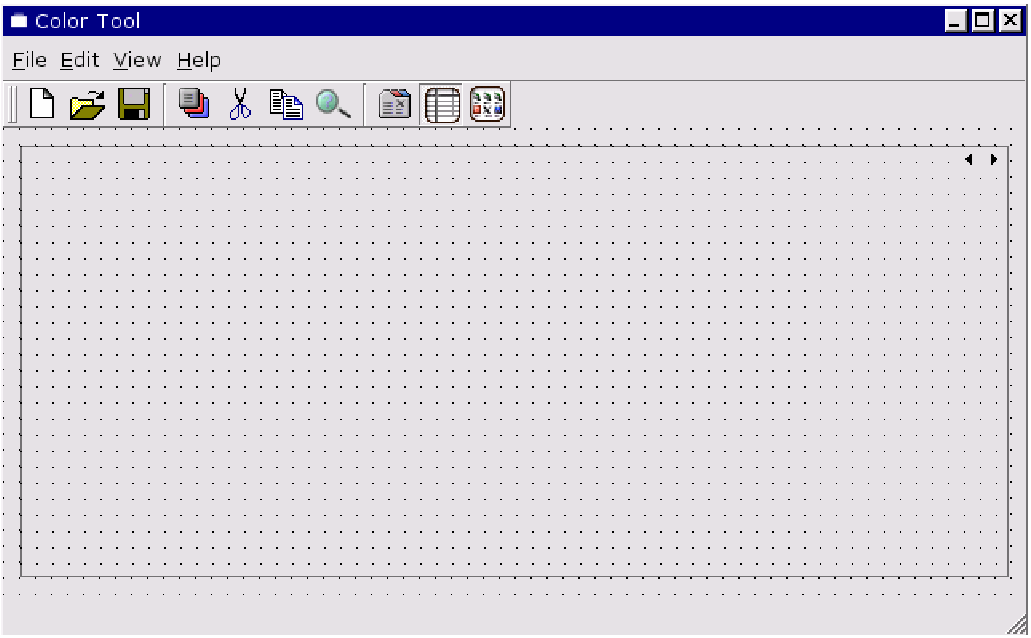


그림 1-33. 창문부품탄창의 확대

Toolbox의 Views단추를 찰칵한다. Table을 선택하고 창문부품탄창의 거의 중간을 찰칵한다. 표의 name속성을 colorTable로, numRows속성을 0으로, readOnly속성을 True로 변경한다.

하나의 창문부품을 오른쪽 단추로 찰칵하여 그 문맥차림표를 펼치면 대부분의 경우에 첫 항목이 Edit선택으로 된다. Table창문부품은 이 점에서 다름이 없고 Edit선택은 행과 렬에 따라 제목들을 변경할수 있는 대화칸을 만든다.

표를 오른쪽 단추로 찰칵하고 Edit…를 선택하여 Edit Table대화칸을 호출한다. 1렬의 Label을 Name으로 변경한다. Columns목록의 2를 선택하여 2렬의 표식자를 Label행편집칸에 표시한다. 2렬의 표식자를 Hex로 변경한다. 같은 방법으로 3렬의 표식자를 Web로 변경한다. OK를 찰칵하여 대화칸을 닫는다.

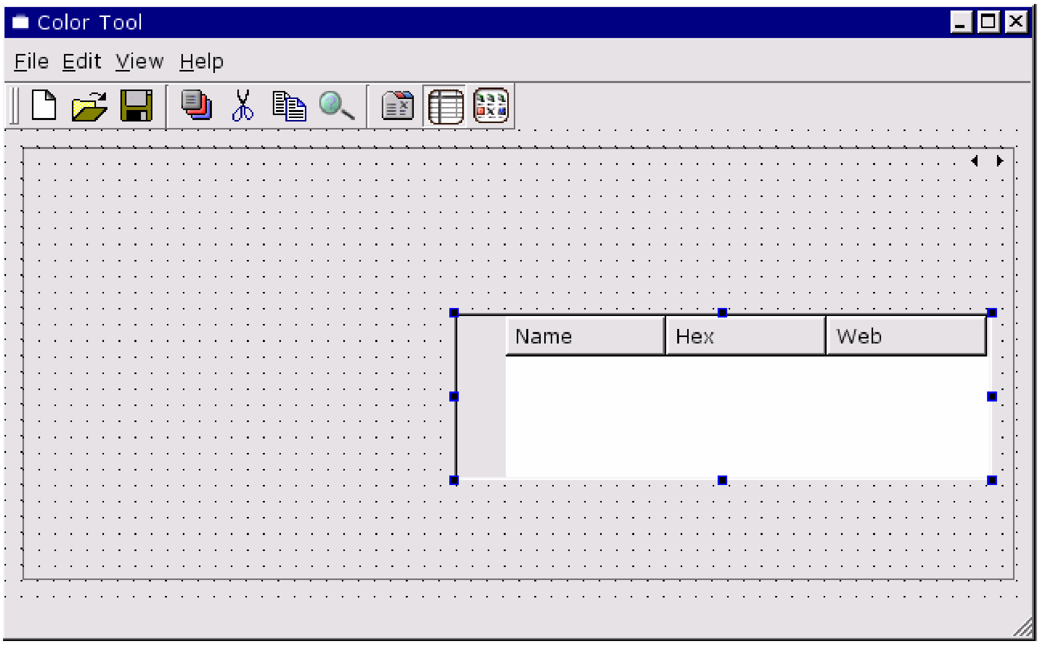


그림 1-34. Table의 추가

창문부품탄창을 찰칵하고 Lay Out Vertically도구띠단추를 찰칵한다. 현재 표는 창문부품탄창에 꼭 들어가고 창문부품탄창으로 크기를 조절한다. (차례로 폼크기를 조절하고 Ctrl+T를 눌러서 미리보기하고 미리보기한 폼의 크기를 조절한다.)

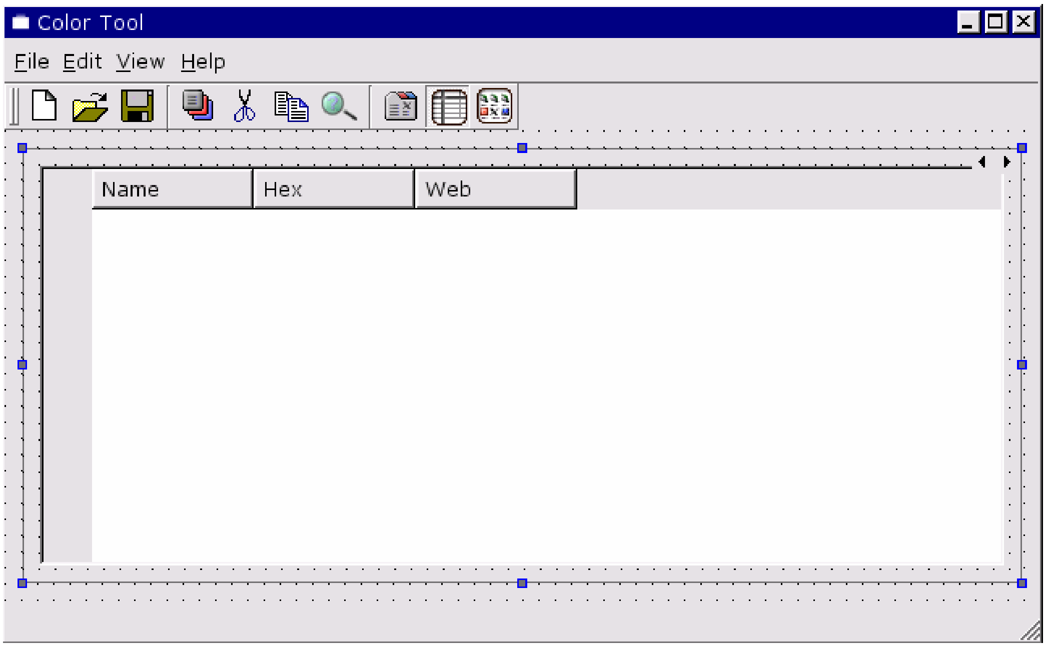


그림 1-35. 수직배치후 tanlePage

Object Explorer에서 WStackPage대상을 선택하고 name을 tablePage로 변경한다.

이제는 다음 페지를 만들 준비가 되였다. 창문부품탄창을 오른쪽 단추로 선택하고 문맥차림표에서 Add Page를 찰칵한다. 표는 표시되지 않았거나 혹은 새로운 창문부품탄창이 표를 포함하는 첫 창문부품탄창페지를 가리운다. Toolbox에서 IconView를 선택하고 창문부품탄창의 거의 중간을 찰칵한다. IconView의 name속성을 colorIconView로, 그 resizeMode속성을 Adjust로 변경한다. 색견본을 작은 칸에 표시하기 위하여 gridX속성을 100으로 변경한다.

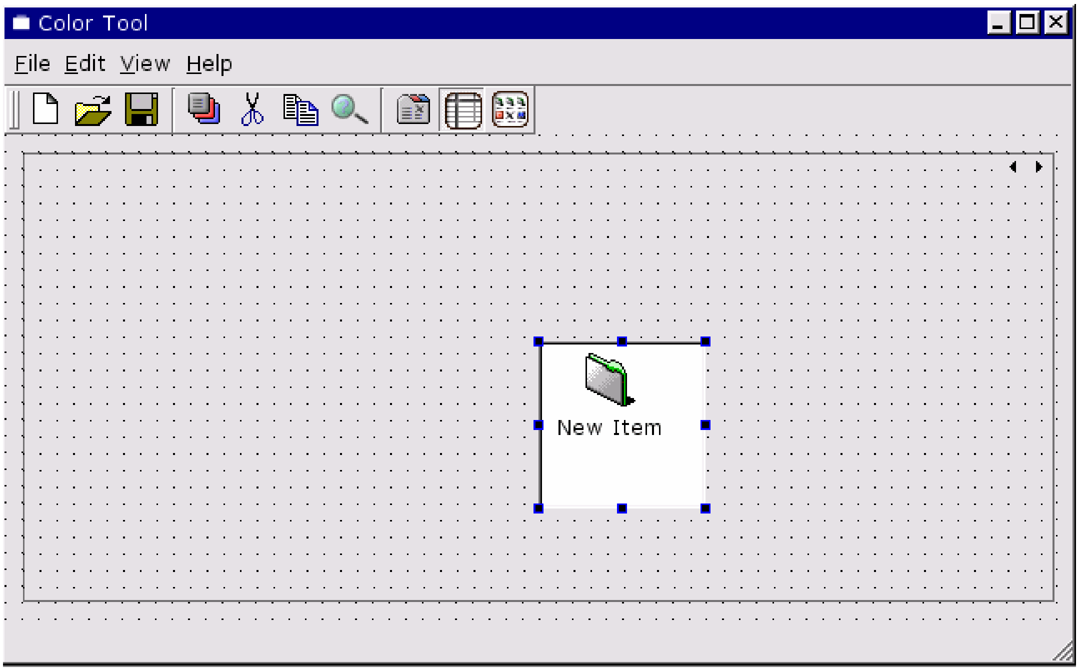


그림 1-36. IconView의 추가

보통 설계할 때 IconView항목들을 만드는것이 효과있지만 이 응용프로그람에는 적합하지 않다. IconView를 오른쪽 단추를 찰칵하면 펼쳐지는 문맥차림표에서 Edit…를 눌러서 Edit IconView대화칸을 열고 Delete Item을 눌러서 기정항목을 삭제하고 OK를 찰칵한다.

창문부품탄창을 선택하고 Lay Out Vertically도구띠단추를 찰칵한다. 지금 그림기호보기는 창문부품탄창에 정확히 맞는다.

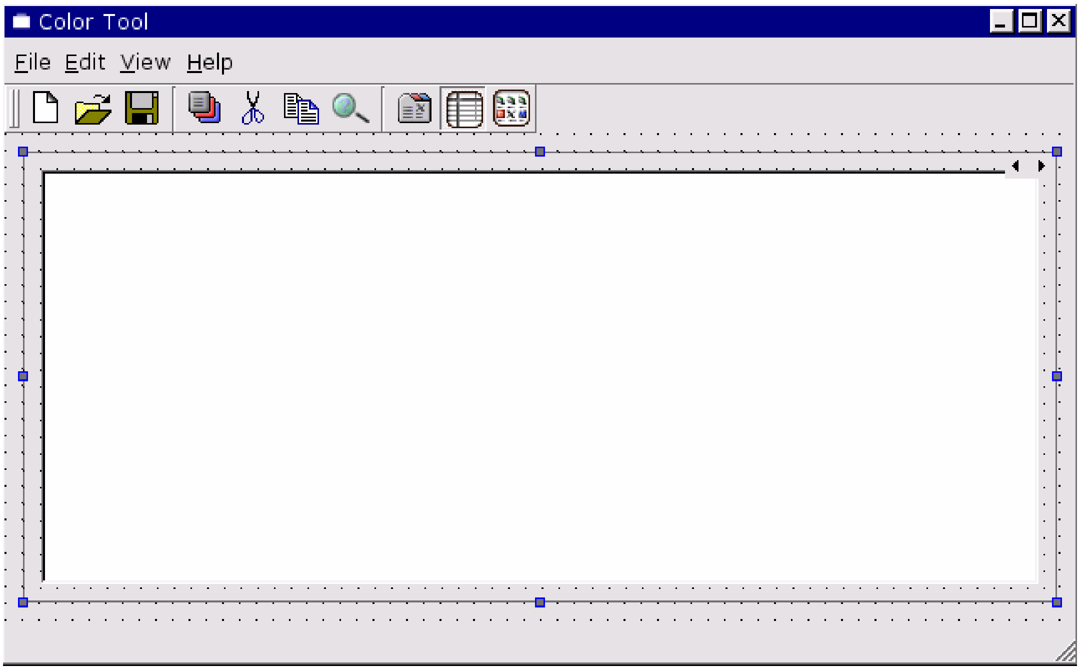


그림 1-37. 수직배치후 iconsPage

Object Explorer에서 WStackPage객체를 선택하고 이름을 iconsPage로 변경한다.

창문부품탄창을 오른쪽 단추로 찰칵하고 Previous Page를 찰칵한다.

이로써 응용프로그람의 기본창문을 위한 사용자대면부설계를 끝낸다. 폼을 미리본다면 View차림표선택과 도구띠단추들이 효과가 없다. 이것은 이 차림표선택들과 도구띠단추들에 의하여 작용들이 발생할 때 실행해야 할 코드를 쓰지 않았기때문이다. 다음에 필요한 코드를 쓴다.

6. 코드작성

Qt Designer로 설계한 코드를 쓰는 2가지 수법이 있다. 원시적인 수법은 이미 만든 매개 폼의 파생클라스를 만들고 파생클라스에 모든 코드를 넣는것이다. Qt 3.0으로부터 Qt Designer는 한가지 방법을 제공한다. Qt Designer에서 코드편집기를 사용하여 코드를 직접 쓸수 있다(4절). 이 실레에서는 Qt Designer에서 모든 코드를 쓴다(파생클라스수법도 있는데 그 실레는 5절을 참고).

코드작성을 시작하기전에 폼변수들을 만들어야 한다. 실레로 보기가 갱신을 요구하는가 하는것을 추적하여야 한다. 사용자가 다른 보기에서 새로운 색모임을 적재하거나 색들을 추가, 삭제하였기때문이다.

1) 성원변수의 추가

Object Explorer의 Members타브를 선택한다. Class Variables(바닥쪽으로)를 오른쪽 단추로 찰칵하고 Edit를 선택한다. Edit Class Variables대화칸이 펼쳐진다. Add단추를 찰칵하고 QMap<QString,QColor> m_colors라고 입력한다. 이 매프를 사용하

여 사용자의 색이름을 색과 련결한다. 다시 Add단추를 찰칵하고 bool m_changed라고 입력한다. 이 변수를 사용하여 자료가 변경되였는가 아닌가를 추적한다. 이것은 가령 완료하거나 새 파일을 열 때 보관하지 않은 자료를 보관하겠는가 하는 질문을 사용자에게 제시한다.

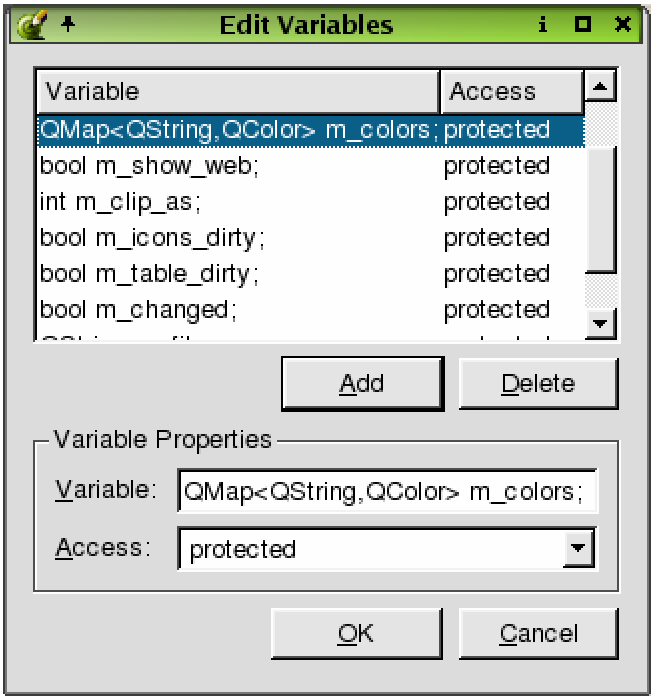


그림 1-38. 성원변수의 추가

갈은 방법으로 QString m_filename을 추가하여 사용자가 연 파일들을 추적할수 있다. bool m_table_dirty과 bool m_icons_dirty를 추가한다. 표를 표시하고있을 때 사용자가 색을 추가하면 그림기호들을 변경된것으로 표식함으로써 사용자가 그림기호보기를 변경한다면 그림기호보기가 갱신되게 할수 있다. bool m_show_web를 추가하고 이것을 사용하여 표에 색이 웨브색인가 아닌가를 가리키는 란을 요구하는가 아닌가를 기록한다. int m_clip_as를 추가하고 이것을 사용하여 사용자가 File|Copy를 선택하여 오려둠판에 복사하도록 할수 있다. QClipboard *clipboard를 추가하여 대역오려둠판에로의 지적자를 보관할수 있다. 끝으로 QStringList m_comments를 추가한다. 이것은 색파일들을 적재하고 보관하는데 사용되며 후에 설명한다.

현재 다음과 같은 변수들이 있다.

QMap<QString,QColor> m_colors;

bool m_changed;

QString m_filename;

bool m_table_dirty;

bool m_icons_dirty;

bool m_show_web;

int m_clip_as;

QClipboard *clipboard;

QStringList m_comments;

Enter를 눌러서 마지막 변수를 확인하고 OK를 찰칵하여 대화칸을 닫는다. 현재 모든 변수들이 Object Explorer의 Members타브에 표시된다.

2) 앞방향선언의 추가

이미 만든 일부 변수들은 앞방향선언을 요구하는 클라스들의 객체이다. Object Explorer의 Members타브에서 Forward Declarations를 오른쪽 단추로 찰칵하고 Edit를 선택한다. 그러면 Edit Forward Declarations대화칸이 펼쳐진다. 이 대화칸은 방금 리용한 Edit Class Variables대화칸와 같은 방법으로 작업한다. 다음의 앞방향선언을 추가한다. 즉 class QString;와 class QColor를 추가한다.. 대화칸을 닫으면 앞방향선언들이 Object Explorer에 표시된다.

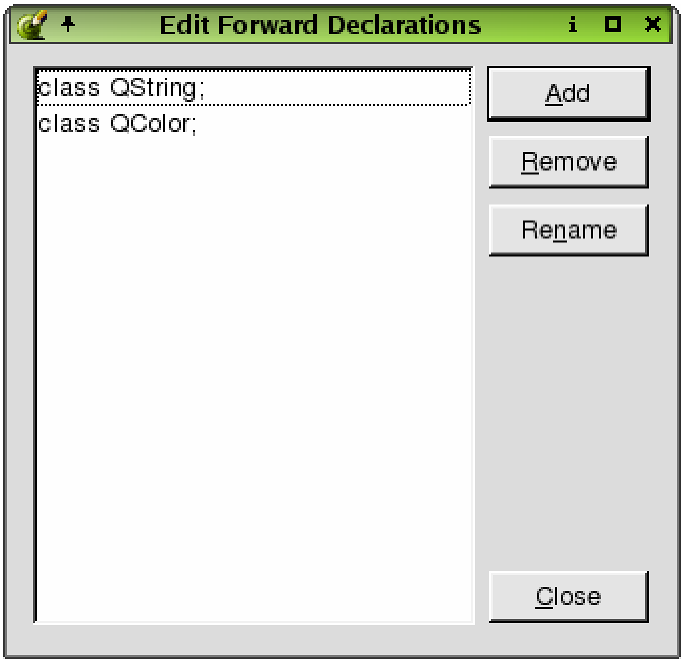


그림 1-39. 앞방향선언의 편집

결과 다음과 같은 앞방향선언이 추가되게 되였다.

class QString;

class QColor;

3) 머리부의 추가

또한 폼은 머리부파일들을 요구한다. 머리부파일들을 선언이나 실현에 추가할수 있다. Includes(in Implementation)을 오른쪽 단추로 찰칵하고 Edit를 누른다. 이때 열리는 대화칸을 통하여 qcolor.h와 qstring.h을 입력한다. 오려둠판을 사용하려면 QApplication을 거쳐서 대역오려둠판을 호출해야 하므로 qapplication.h와 qclipboard.h를 추가한다. 또한 그리기조작(례하면 색견본)을 하기 위하여 qpainter.h도 추가하고 대화칸을 닫는다.

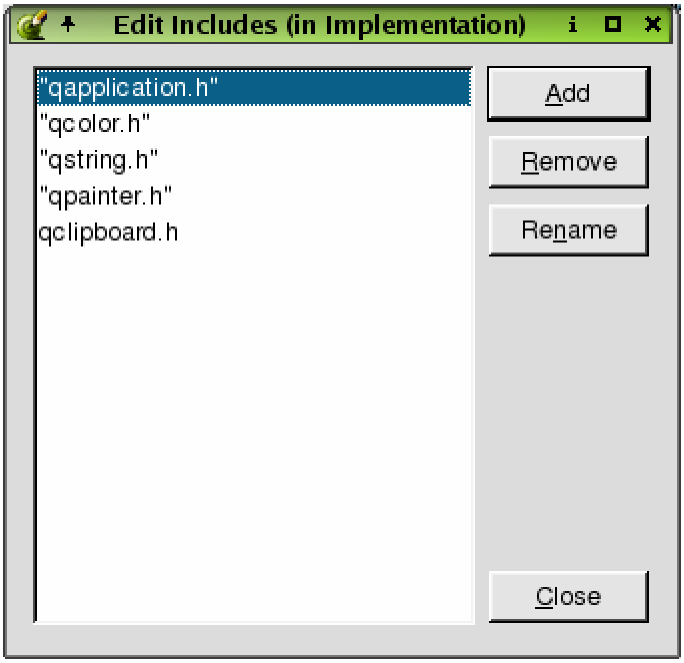


그림 1-40. 머리부의 추가

머리부파일을 입력할 때 2중인용표나 각괄호를 포함할수 있다. 괄호를 사용하지 않으면 Qt Designer는 자동적으로 2중인용표에 넣는다.

실현파일에 다음의 머리부를 추가한다.

"qcolor.h"

"qstring.h"

"qapplication.h"

"qclipboard.h"

"qpainter.h"

4) 신호와 처리부의 련결

대부분의 신호-처리부련결은 기본폼을 작성할 때 기본창문위자드에 의하여 자동적으로 창조된다. 그다음 새로운 작용들을 추가한다. 작용들의 동작에 대한 코드를 쓸수 있도록 그것들이 처리부에 련결되여있어야 한다.

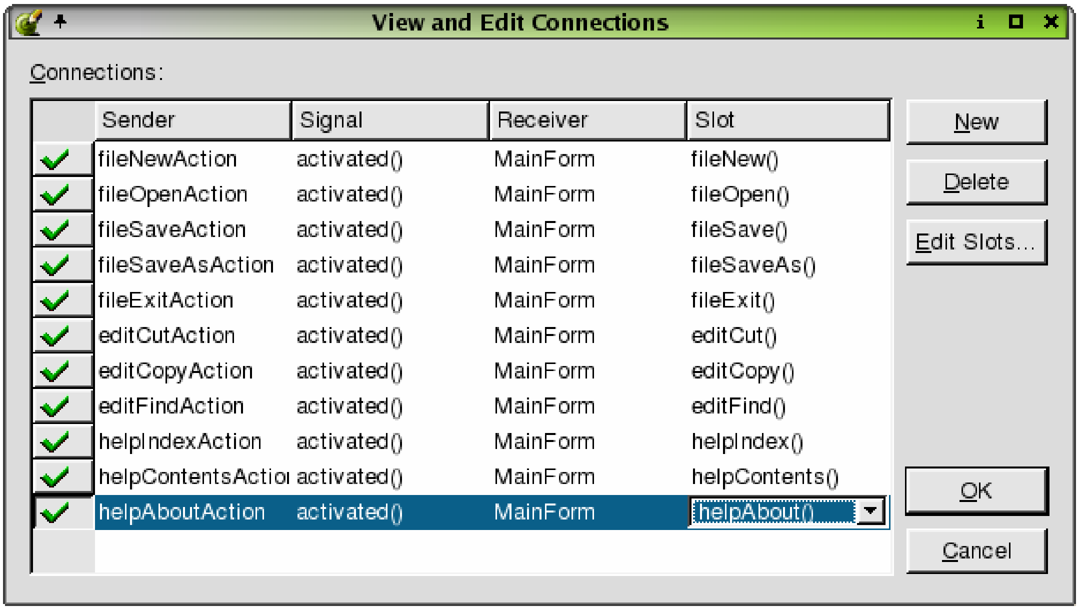


그림 1-41. 신호와 처리부의 련결(1)

- 신호-처리부련결의 작성

Edit|Connections을 찰칵하여 View and Edit Connections대화칸을 펼친다.

보통 이 대화칸의 사용은 같은 본보기를 따른다. New를 찰칵하여 새 련결을 생성하고 Sender창문부품, 송신기의 Signal, Receiver(보통 폼)을 선택한다. 미리 정의된 처리부를 사용하려고 한다면 그 처리부를 선택하고 그렇지 않으면 Edit Slots...를 선택하여 새 처리부를 만들고 새로 만든 처리부를 선택한다. (련결을 선택하여 끌고가는 이전의 방법이 여전히 지원되지만 새 방법은 한번의 이행으로 여러개의 련결을 만들 때 특별히 더 빠르고 더 간단하다.)

상태띠를 갱신하여 사용자가 색정보를 볼수 있게 하려고 한다. Edit|Connections을 찰칵하여 View and Edit Connections대화칸을 열고 New를 눌러서 새 련결을 만든다. Sender를 colorTable로, Signal을 currentChanged(int,int)로, Receiver를 MainForm으로 변경한다.

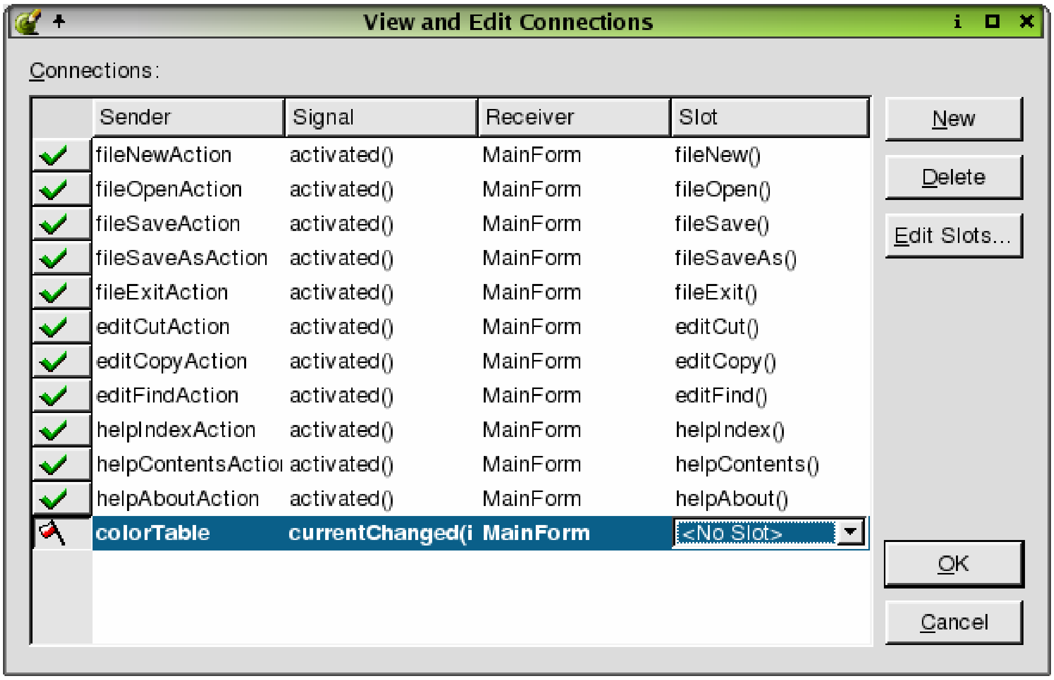


그림 1-42. 신호와 처리부의 련결(2)

아직 만들지 않은 자체의 전용처리부에 련결하려고 한다. Edit Slots단추를 찰칵하여 Edit Functions대화칸을 열고 New Function를 찰칵하고 처리부이름을 changedTableColor(int,int)로 변경한다. OK를 찰칵하여 대화칸을 닫는다.

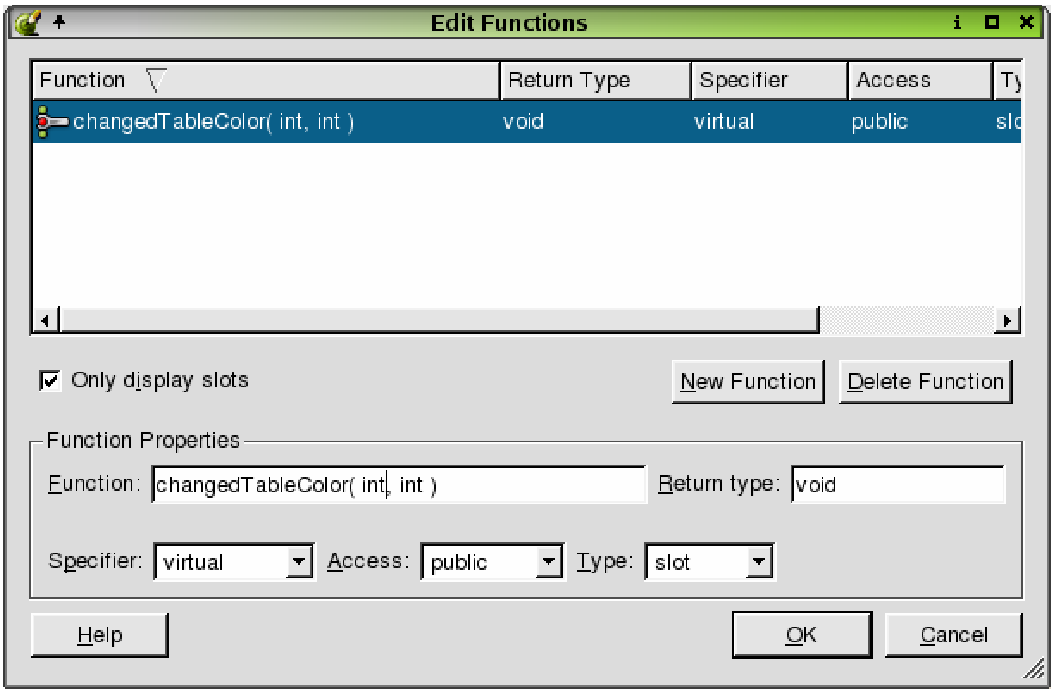


그림 1-43. changedTableColor()함수의 편집

그다음 View and Edit Connections대화칸에서 Slot를 새로 만든 changedTableColor(int,int)로 변경한다.

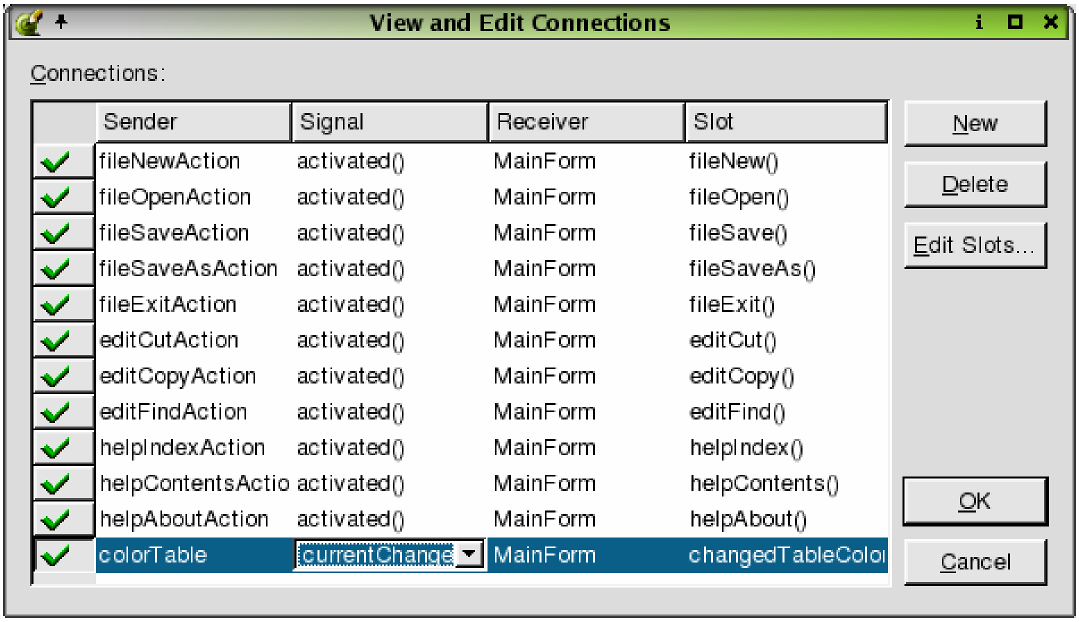


그림 1-44. currentChanged()신호와 changedTableColor()처리부의 련결

New를 찰칵하여 새 련결을 만들고 Sender를 colorIconView로, Signal을 currentChanged(QIconViewItem*)로, Receiver를 MainForm로 변경한다. Edit Slots…단추를 찰칵하여 Edit Functions대화칸을 펼치고 New Function을 찰칵하고 처리부이름을 changedIconColor(QIconViewItem*)로 변경한다. OK를 찰칵하여 대화칸을 닫는다. View and Edit Connections대화칸에서 Slot를 changedIconColor(QIconViewItem*)로 변경한다.

이제는 changedTableColor()와 changedIconColor()처리부들을 실현하여 상태띠를 현재색에 대한 설명으로 갱신할수 있다.

또한 사용자가 보기를 변경할 때 보기에 보여준 색이 정확하다는것을 담보하려고 한다. 실례로 사용자가 표의 색을 삭제하여 그림기호보기를 변경하였다면 그림기호보기가 삭제된 색을 표시하지 않도록 담보해야 한다.

New를 찰칵하여 새 련결을 만들고 Sender를 colorWidgetStack로, Signal을 aboutToShow(int)로, Receiver를 MainForm으로 변경한다. aboutToShow()라는 새 처리부를 만들고 이것을 창문부품탄창의 aboutToShow(int)신호를 련결하는 처리부로 만든다. 신호는 표시하려는 창문부품의 ID를 포함하지만 파라메터를 가지지 않는 처리부를 창조하므로 그것을 요구하지 않는다.

하나의 중요한 기능은 사용자가 보기들사이를 절환하게 하는것이다. 매개 보기작용들을 따로따로 결합할수 있지만 모두를 작용그룹으로 결합하면 더 편리하고 확장하기 쉽다.

Sender로서 viewActionGroup를 가지는 새 련결을 만들고 Signal을 selected(QAction*)로, Receiver를 MainForm으로 변경한다. changeView(QAction*)라는 처리부를 만들고 이것을 신호가 련결하는 처리부로 만든다.

OK를 찰칵하여 View and Edit Connections대화칸을 연다. 이제는 코드를 쓸 준비가 되였다.

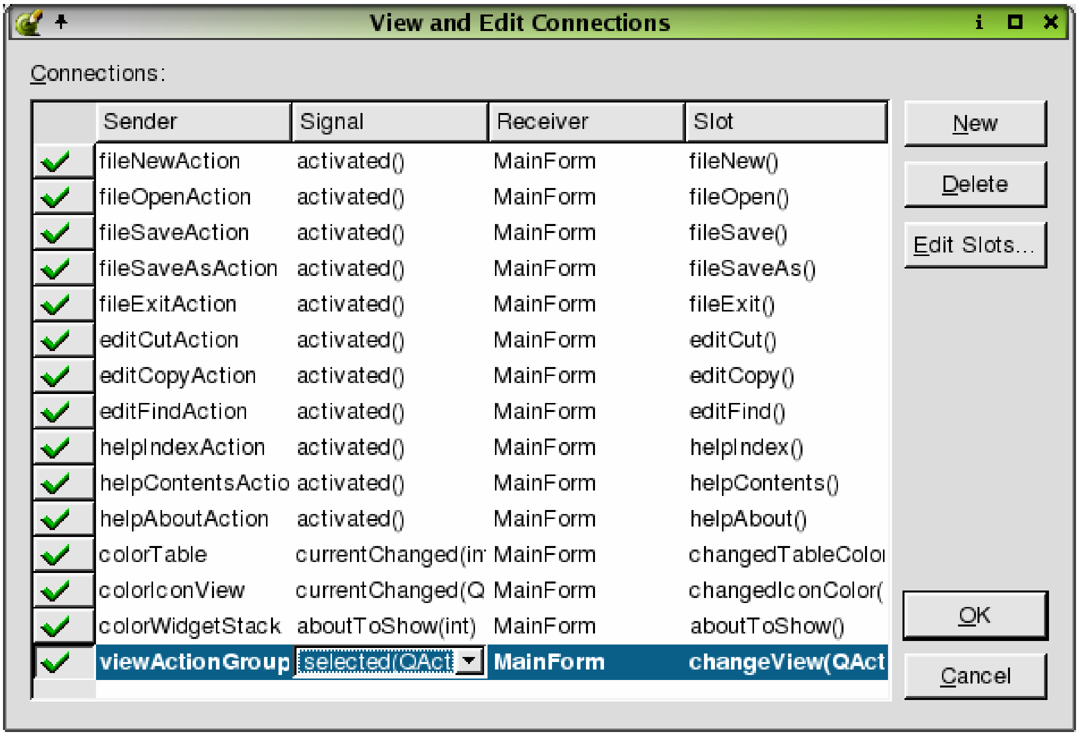


그림 1-45. 신호와 처리부의 련결

5) 설정코드의 편집

응용프로그람에 포함할 코드는 매우 많지만 많이 입력해야 한다는것을 의미하지 않는다.

Project Overview창문에서 mainform.ui.h를 찰칵한다. 빈 처리부를 표시하는 코드편집기창문이 나타난다.

- 상수의 추가

```
const int CLIP_AS_HEX = 0;
const int CLIP_AS_NAME = 1;
const int CLIP_AS_RGB = 2;
const int COL_NAME = 0;
const int COL_HEX = 1;
const int COL_WEB = 2;
const QString WINDOWS_REGISTRY = "/QtExamples";
```

```
const QString APP_KEY = "/ColorTool/";
```

2보다 CLIP_AS_RGB를 기억하기 쉬우므로 폼에 사용할수 있는 상수들을 정의한다. 사용자의 선택을 적재하고 보관할 때 2개의 QString을 QSettings에서 사용한다. 그것들은 loadOptions()와 saveOptions()에서 설명한다. .ui.h파일에 상수선언과 #include를 비롯하여 유효한 C++를 삽입할수 있다.

파생클라스를 만들지 않으므로 구성시에 실행되는 코드를 요구한다면 init()함수를 만들어야 한다. 이것은 폼구성자의 끝에서 호출된다.

- init()

```
void MainForm::init()
{
    clipboard = QApplication::clipboard();
    if ( clipboard->supportsSelection() )
        clipboard->setSelectionMode( TRUE );

    findForm = 0;
    loadSettings();
    m_filename = "";
    m_changed = FALSE;
    m_table_dirty = TRUE;
    m_icons_dirty = TRUE;
    clearData( TRUE );
}
```

처음에 대역오려둠판객체에로의 지적자를 얻는다. setSelectionMode()호출은 오려둠판이 모든 가동환경들에서 기대한대로 작업한다는것을 담보한다. findForm과 loadSettings()행들은 후에 학습한다. 코드를 입력하고있다면 현재는 그것들을 설명문으로 한다. 사용자가 파일을 열지 않았으므로 파일이름을 비운다. 아직 어떤 변경도 없으므로 변경을 거짓으로 설정한다. 그러나 표와 그림기호보기를 직접 그리려고 하지 않으므로 그것들을 모두 변경한것(dirty)으로서 표식한다. 다음번에 쓰려는 clearData()함수를 호출한다. 이 함수는 모든 색자료를 지우고 TRUE로 호출되면 기정값을 가지는 새로운 색들을 만든다.

- clearData()

```
void MainForm::clearData( bool fillWithDefaults )
{
    setCaption( "Color Tool" );
```

```
    m_colors.clear();
    m_comments.clear();

    if ( fillWithDefaults ) {
      m_colors["black"] = Qt::black;
      m_colors["blue"] = Qt::blue;
      m_colors["cyan"] = Qt::cyan;
      m_colors["darkblue"] = Qt::darkBlue;
      m_colors["darkcyan"] = Qt::darkCyan;
      m_colors["darkgray"] = Qt::darkGray;
      m_colors["darkgreen"] = Qt::darkGreen;
      m_colors["darkmagenta"] = Qt::darkMagenta;
      m_colors["darkred"] = Qt::darkRed;
      m_colors["darkyellow"] = Qt::darkYellow;
      m_colors["gray"] = Qt::gray;
      m_colors["green"] = Qt::green;
      m_colors["lightgray"] = Qt::lightGray;
      m_colors["magenta"] = Qt::magenta;
      m_colors["red"] = Qt::red;
      m_colors["white"] = Qt::white;
      m_colors["yellow"] = Qt::yellow;
    }
    populate();
}
```

이 함수는 응용프로그람을 기동할 때와 사용자가 새 파일을 만들거나 현존파일을 적재할 때 사용된다. 이때 자료를 지우고 선택적으로 기정색들을 삽입한다. 파일들을 적재하고 보관할 때 파일이름을 제목에 추가하므로 응용프로그람의 제목을 설정한다. 또한 삭제할 때에는 제목으로부터 파일이름을 지워야 한다. 색매프와 설명문자렬목록을 지우고 선택적으로 색략도를 표준색으로 채운다. 끝으로 표와 그림기호보기에 자료를 채우는 populate()를 호출한다.

- populate()

```
void MainForm::populate()
{
```

```
if ( m_table_dirty ) {
    for ( int r = 0; r < colorTable->numRows(); ++r ) {
        for ( int c = 0; c < colorTable->numCols(); ++c ) {
            colorTable->clearCell( r, c );
        }
    }

    colorTable->setNumRows( m_colors.count() );
    if ( ! m_colors.isEmpty() ) {
        QPixmap pixmap( 22, 22 );
        int row = 0;
        QMap<QString,QColor>::ConstIterator it;
        for ( it = m_colors.constBegin(); it != m_colors.constEnd(); ++it ) {
            QColor color = it.data();
            pixmap.fill( color );
            colorTable->setText( row, COL_NAME, it.key() );
            colorTable->setPixmap( row, COL_NAME, pixmap );
            colorTable->setText( row, COL_HEX, color.name().upper() );
            if ( m_show_web ) {
                QCheckTableItem *item = new QCheckTableItem(colorTable,"" );
                item->setChecked( isWebColor( color ) );
                colorTable->setItem( row, COL_WEB, item );
            }
            row++;
        }
        colorTable->setCurrentCell( 0, 0 );
    }
    colorTable->adjustColumn( COL_NAME );
    colorTable->adjustColumn( COL_HEX );
    if ( m_show_web ) {
        colorTable->showColumn( COL_WEB );
        colorTable->adjustColumn( COL_WEB );
    }
    else
```

```
        colorTable->hideColumn( COL_WEB );
    m_table_dirty = FALSE;
}

if ( m_icons_dirty ) {
    colorIconView->clear();

    QMap<QString,QColor>::ConstIterator it;
    for ( it = m_colors.constBegin(); it != m_colors.constEnd(); ++it )
        (void) new QIconViewItem( colorIconView, it.key(),
                        colorSwatch( it.data() ) );
    m_icons_dirty = FALSE;
}
}
```

이 함수는 응용프로그람의 심장부이다. 이것은 사용자에게 자료를 시각적으로 제시한다. 표가 갱신되였으면(실례로 사용자가 그림기호보기에서 색을 추가 혹은 삭제하거나 색파일을 열었다면) 표를 구성한다. 각 세포의 내용을 삭제하는것으로 시작한다. 다음에 색략도의 색수와 같게 행수를 변경한다. 매개 색에 대하여 색을 보여주는 작은 장방형을 표시하기 위하여 요구되는 크기의 픽스매프를 만든다.

이제 색매프용 반복자를 생성하고 매개 색을 순환한다. 색매프는 사용자의 색이름과 값으로서 QColor실례를 가진다. 색을 얻어서 그 색으로 픽스매프를 채운다. 그다음 색이름(it.key())과 그 색으로 채운 픽스매프를 얻기 위하여 Name칸(COL_NAME칸)을 설정한다. QColor의 name()함수는 색의 16진표시인 문자렬 례하면 #12AB2F를 돌려준다. 이것을 얻어서 둘째칸(Hex)을 이 값으로 설정한다.

사용자가 색이 웨브색인가 알아보려면 QCheckTableItem을 생성하고 그것이 웨브색인가 검사한다. (간단히 isWebColor()를 호출한다.) 그다음 이 QCheckTableItem을 Web란에 삽입한다.

표를 구성하고 adjustColumn()를 호출하여 매개 칸이 그 창문부품항목에 충분한 폭을 가지며 사용자의 선택에 따라 Web칸을 표시 혹은 숨기도록 해야 한다.

끝으로 m_table_dirty를 FALSE로 설정하여 갱신한다.

그림기호보기가 갱신되였으면 현존자료를 지운다. 그다음 색매프의 매개 색을 순환한다. 매개 색에 대하여 새로운 QIconViewItem을 생성한다. 사용자의 색이름을 가지는 항목을 표식하고 련관된 색으로 된 픽스매프(colorSwatch()에 의해 간단히 생성)를 제공한다. 끝으로 m_icons_dirty를 FALSE로 설정하여 갱신한다.

- isWebColor()

```
bool MainForm::isWebColor( QColor color )
{
    int r = color.red();
    int g = color.green();
    int b = color.blue();

    return ( ( r ==   0 || r ==  51 || r == 102 ||
               r == 153 || r == 204 || r == 255 ) &&
             ( g ==   0 || g ==  51 || g == 102 ||
               g == 153 || g == 204 || g == 255 ) &&
             ( b ==   0 || b ==  51 || b == 102 ||
               b == 153 || b == 204 || b == 255 ) );
}
```

216웨브색은 RGB값들이 모두 모임(0, 51, 102, 153, 204, 255)안에 있는 색들이다.

- colorSwatch()

```
QPixmap MainForm::colorSwatch( const QColor color )
{
    QPixmap pixmap( 80, 80 );
    pixmap.fill( white );
    QPainter painter;
    painter.begin( &pixmap );
    painter.setPen( NoPen );
    painter.setBrush( color );
    painter.drawEllipse( 0, 0, 80, 80 );
    painter.end();
    return pixmap;
}
```

적당한 크기의 픽스매프를 생성하고 그것을 흰색으로 채운다. 그다음 픽스매프를 그리는데 사용하는 QPainter를 생성하다. 그리려는 도형주위에 륜곽선이 필요없으므로 펜을 요구하지 않는다. 80×80화소의 정방형안에 타원(즉 원)을 그린다. 결과 픽스매프를 돌려준다.

6) main.cpp의 작성

일부 코드를 입력하였으므로 응용프로그람을 건설실행하여 그 동작을 볼수 있다. 그러기 위하여서는 main()함수를 만들어야 한다. 일반적으로 Qt에서는 main()함수용으로 작은 main.cpp파일을 만든다. Qt Designer에 의해 이 파일을 만들수 있다.

・File|New를 눌러서 New File대화칸을 펼친다. C++ Main-File을 선택하고 OK를 찰칵한다. Configure Main-File대화칸이 열리고 프로젝트안의 모든 폼들이 렬거된다. 하나의 폼 MainForm만 만들었으므로 그것이 이미 선택되여있다. OK를 찰칵하여 MainForm을 적재하는 main.cpp파일을 생성한다.

```
#include <qapplication.h>
#include "mainform.h"

int main( int argc, char ** argv )
{
    QApplication a( argc, argv );
    MainForm *w = new MainForm;
    w->show();
    return a.exec();
}
```

Qt Designer가 main.cpp파일을 생성할 때 다음의 행을 포함한다.

```
a.connect( &a, SIGNAL( lastWindowClosed() ), &a, SLOT( quit() ) );
```

이 코드를 그대로 놔두면 사용자는 기본창문의 닫기(X)단추를 찰칵하여 자체의 완료코드를 무시할수 있다. 사용자에게 보관되지 않은 변경을 보관하는 선택을 줄수 있다. 이것을 달성하기 위하여 그 련결을 삭제하고 응용프로그람을 닫으려는 시도를 차단하고 fileExit()함수를 호출하는 새로운 처리부 closeEvent()를 추가한다.

・Project Overview창문에서 main.cpp를 찰칵한다. 편집창문에 파일이 나타난다. 련결행을 삭제한다.

・Project Overview창문에서 mainform.ui.h를 찰칵한다.(우선 mainform.ui을 선택하여 mainform.ui.h을 표시한다). Object Explorer의 Members목록(Slots, public)에서 fileExit()를 오른쪽 단추로 선택하고 Goto Implementation을 찰칵한다. fileExit()처리부우에 다음의 처리부를 추가한다.

```
void MainForm::closeEvent( QCloseEvent * )
{
    fileExit();
}
```

그러면 사용자가 응용프로그람을 닫으려고 할 때 fileExit()처리부가 호출된다. fileExit()처리부의 코드를 작성하자.

```
void MainForm::fileExit()
{
      QApplication::exit( 0 );
}
```

이것은 응용프로그람이 정확히 완료되는것을 담보한다. 후에 이 함수를 갱신하여 사용자에게 보관되지 않은 변경을 보관할 기회를 주게 한다.

7) 건설과 실행

이리하여 응용프로그람의 코드와 main()함수를 포함하는 main.cpp를 가지게 되였으므로 응용프로그람을 콤파일, 련결, 실행할수 있다.

File|Save를 찰칵하여 모든 작업이 디스크에 보관되도록 한다. 말단창문(실례로 xterm이나 DOS창문)을 열고 등록부를 colortool프로젝트를 보관하려는 곳으로 변경하고 qmake를 실행하여 Makefile을 만든다.

```
qmake -o Makefile colortool.pro
```

다음 프로젝트를 만든다. (Windows에서 nmake, 다른 가동환경에서 make를 실행한다.) init()함수에서 설명문으로 한 findForm과 loadSettings행들을 제공하여 프로그람을 건설해야 한다.

make가 완료하면 프로그람을 실행한다. 아직 보기를 변경할수 없지만 기정색모임을 생성한다. 닫기(X)단추 혹은 File|Exit를 눌러서 응용프로그람을 완료할수 있다.

8) 상태띠의 갱신코드편집

상태띠에 현재색에 대한 정보를 표시하고 사용자가 보기를 변경하거나 색파일에 적재할 때 련관된 보기가 갱신된다는것을 담보하려고 한다.

- aboutToShow()

```
void MainForm::aboutToShow()
{
   populate();
}
```

populate()처리부를 만들고 그것을 직접 호출한다. 우리는 간접적방법을 사용한다. 왜냐하면 그것이 더 명백하고 실제 응용프로그람에서 이 처리부에서 진행하는 경우가 아마 더 많기때문이다.

- changedTableColor()

```
void MainForm::changedTableColor( int row, int )
{
```

```
    changedColor( colorTable->text( row, COL_NAME ) );
}
```

사용자가 표보기에서 이동하거나 마우스를 눌렀는가를 알아내도록 하기 위하여 이 처리부에 련결한다. 현재색의 이름으로 changedColor()함수를 호출한다. column인수를 고려하지 않으므로 그것을 남겨둔다. changedTableColor파라메터의 이름을 "int row"로 하였다.

- changedIconColor()

```
void MainForm::changedIconColor( QIconViewItem *item )
{
    changedColor( item->text() );
}
```

이 처리부는 우의 changedTableColor()와 같은 목적으로 련결된다. 또한 현재색의 이름으로 changedColor()를 호출한다. (코드를 자르기하여 붙이기하는 경우에 QIconViewItem파라메터의 이름을 item으로 하였다는것을 잊어서는 안된다.)

- changedColor()

간단히 코드를 Qt Designer의 코드편집기에 입력하면 Object Explorer의 Members타브(Functions, public아래)에 자동적으로 표시된다.

기정으로 코드편집기에 직접 입력한 함수는 공개함수로 된다. 이것을 변경하려면 Object Explorer의 Members목록에서 함수이름을 오른쪽 단추로 찰칵하고 Properties를 선택하여 Edit Functions대화칸을 펼친 다음 처리부로의 변경을 비롯하여 함수의 여러가지 속성을 변경할수 있다.

```
void MainForm::changedColor( const QString& name )
{
    QColor color = m_colors[name];
    int r = color.red();
    int g = color.green();
    int b = color.blue();
    statusBar()->message(QString("%1 \"%2\" (%3,%4,%5)%6 {%7 %8 %9}" ).
                    arg( name ).
                    arg( color.name().upper() ).
                    arg( r ).arg( g ).arg( b ).
                    arg( isWebColor( color ) ? " web" : "" ).
                    arg( r / 255.0, 1, 'f', 3 ).
                    arg( g / 255.0, 1, 'f', 3 ).
```

```
                    arg( b / 255.0, 1, 'f', 3 )
                    );
}
```

이 함수는 색매프에서 색이름을 찾고 이름이 제공하는 색을 얻는다. 그다음 이름과 16진값, 색이 웨브색인가 아닌가를 상태띠에 표시한다.

QMainWindow는 실제로 상태띠를 사용하다면 그것을 만든다. 우리가 지금까지 그것을 사용하지 않았으므로 문제가 없었지만 콤파일하려고 하면 현재 상태띠를 사용하고있으나 련관된 머리부를 선언하지 않았으므로 오유가 발생한다. Object Explorer의 Members타브를 누르고 qstatusbar.h를 Includes (In Implementation)절에 추가한다. Includes (In Implementation)을 오른쪽 단추로 찰칵하고 New를 누르고 qstatusbar.h를 입력하고 Enter를 누른다.

이제는 머리부(실현)에 다음의 선언을 추가해야 한다.

- "qstatusbar.h"

Ctrl+S를 눌러서 응용프로그람을 보관하고 구축하고 실행한다. 각이한 색으로 이동하면서 상태띠에서 색을 설명하고있는가를 확인한다. (건설되지 않으면 3절. 6을 참고.)

- 보기의 변경

지금까지는 보기를 절환하는 코드가 없으므로 작용에서 그림기호보기를 확인할수 없다. 이제 이 문제를 해결하자.

사용자가 보기도구띠단추 혹은 차림표선택을 눌렀을 때 호출되는 changeView() 처리부를 이미 만들었으므로 코드를 써야 한다.

```
void MainForm::changeView(QAction* action)
{
    if ( action == viewTableAction )
        colorWidgetStack->raiseWidget( tablePage );
    else
        colorWidgetStack->raiseWidget( iconsPage );
}
```

9) 파일조종코드의 편집

X협회가 이미 색과 색이름을 련결하기 위한 파일형식을 정의하였으므로 응용프로그람용으로 따로 만들지 않고 그 파일형식을 사용한다. 이것은 rgb.txt를 읽고쓸수 있게 하며 이 형식이 다른 사용자들에게 친숙해질수 있는 우점을 가지고있다.

- fileNew()

```
void MainForm::fileNew()
{
```

```
    if ( okToClear() ) {
        m_filename = "";
        m_changed = FALSE;
        m_table_dirty = TRUE;
        m_icons_dirty = TRUE;
        clearData( FALSE );
    }
}
```

이 함수는 자료를 적재하거나 보관하지 않고 단순히 현존자료를 지우는것이 좋겠는가를 okToClear()를 호출하여 확인하고 좋으면 폼을 초기화한다.

- okToClear()

새로운 색모임을 생성하거나 현존모임을 적재하기전에 보관되지 않은 변경이 있는가 확인해야 한다. 있다면 사용자에게 그 자료를 보관할 기회를 주어야 한다. 그것을 이 함수가 수행한다.

```
bool MainForm::okToClear()
{
    if ( m_changed ) {
        QString msg;
        if ( m_filename.isEmpty() )
            msg = "Unnamed colors ";
        else
            msg = QString( "Colors '%1'\n" ).arg( m_filename );
        msg += QString( "has been changed." );
        int ans = QMessageBox::information(this,
                "Color Tool -- Unsaved Changes", msg, "&Save", "Cancel",
                "&Abandon", 0, 1 );
        if ( ans == 0 )
            fileSave();
        else if ( ans == 1 )
            return FALSE;
    }
    return TRUE;
}
```

자료가 변경되였으면(m_changed가 TRUE이면) 사용자에게 자료를 보관하는가

아니면 현재 조작을 취소(실례로 새 파일을 적재하지 않거나 새로운 색모임을 생성하지 않는다)하는가, 변경을 무시하고 계속하는가 하는 선택을 제공하는 통보창문을 표시한다. Save단추를 기정단추로 만들고(Enter를 누른다.) Cancel단추를 취소단추(Esc를 누른다.)로 만든다.

QMessageBox를 사용하므로 련관된 머리부를 포함해야 한다. (Includes (in Implementation)를 오른쪽 단추로 찰칵하고 New를 누른다. qmessagebox.h라고 입력하고 Enter를 누른다.)

이제는 실현의 머리부에 다음의 선언을 추가해야 한다.

·"qmessagebox.h"

fileOpen()

```
void MainForm::fileOpen()
{
    if ( ! okToClear() )
        return;

    QString filename = QFileDialog::getOpenFileName(
                        QString::null, "Colors (*.txt)", this,
                        "file open", "Color Tool -- File Open" );
    if ( ! filename.isEmpty() )
        load( filename );
    else
        statusBar()->message( "File Open abandoned", 2000 );
}
```

자료를 지우면 안되는 경우에(즉 사용자가 변경을 보관하지 않았는데 okToClear()에 의하여 펼쳐진 통보창문에서 Cancel을 눌렀을 때) 단순히 되돌아간다. 그렇지 않으면 QFileDialog의 정적함수를 리용하여 사용자로부터 파일이름을 얻은 다음 파일을 적재한다.

QFileDialog를 사용하므로 련관된 머리부를 포함한다.(Includes(in Implementation)를 오른쪽단추를 찰칵하고 New를 선택한다. qfiledialog.h라고 입력하고 Enter를 누른다.)

이제는 실현의 머리부들에 다음의 선언을 추가한다.

"qfiledialog.h"

- load()

```
void MainForm::load( const QString& filename )
{
```

```
    clearData( FALSE );
    m_filename = filename;
    QRegExp regex( "^\\s*(\\d+)\\s+(\\d+)\\s+(\\d+)\\s+(\\S+.*)$" );
    QFile file( filename );
    if ( file.open( IO_ReadOnly ) ) {
      statusBar()->message( QString( "Loading '%1'..." ). arg( filename ) );
      QTextStream stream( &file );
      QString line;
      while ( ! stream.eof() ) {
        line = stream.readLine();
        if ( regex.search( line ) == -1 )
          m_comments += line;
        else
          m_colors[regex.cap( 4 )] = QColor( regex.cap( 1 ).toInt(),
                          regex.cap( 2 ).toInt(), regex.cap( 3 ).toInt() );
      }
      file.close();
      m_filename = filename;
      setCaption( QString( "Color Tool -- %1" ).arg( m_filename ) );
      statusBar()->message( QString( "Loaded '%1'" ). arg( m_filename ), 3000 );
      QWidget *visible = colorWidgetStack->visibleWidget();
      m_icons_dirty = ! ( m_table_dirty = ( visible == tablePage ) );
      populate();
      m_icons_dirty = ! ( m_table_dirty = ( visible != tablePage ) );
      m_changed = FALSE;
    }
    else
      statusBar()->message( QString( "Failed to load '%1'" ).
                    arg( m_filename ), 3000 );
  }
```

새 자료를 적재하기전에 현존자료를 지워야 한다. rgb.txt파일의 형식은 다음과 같다.

RED WHITESPACE GREEN WHITESPACE BLUE WHITESPACE NAME

여기서 RED와 GREEN, BLUE는 0～255범위의 10진수로서 필요한 곳에 공백이 들어있는 3개 문자로 이루어진다. 색들사이의 WHITESPACE는 보통 하나의 공백이고

BLUE와 NAME사이의 WHITESPACE는 2개의 타브이다. NAME은 공백을 포함할수 있다. 실례로

0	191	255	연하늘색
176	48	96	밤색
199	21	133	중간붉은보라색

또한 파일은 설명행을 포함할수 있다. 이것들은 실례에서 '!'로 시작한다.

이 파일들의 구문을 해석하는 수법은 많지만 간단한 정규표현(regex)을 선택하였다. 정규표현은 형식요구보다도 입력에서 공백에 관하여 더욱 《자유적》이다.

한 행이 정규표현과 일치하면 QMap형의 m_colors에 새 항목을 생성하고 그 본문을 색이름(regex.cap(4))으로, 그 값을 새로운 적록청값들로부터 생성된 QColor로 설정한다. 정규표현과 일치하지 않는 행들은 설명문으로 취급되고 m_comments문자렬목록에 보관된다. (파일을 보관할 때 파일중간에서 나타나더라도 모든 설명문을 쓴다.)

일단 m_colors매프를 구성하면 현재 보이는 보기를 변경된것으로 표식하고 populate()를 호출하여 그것을 갱신한다. 그다음 보이는 보기를 변경되지 않은것으로 표식하고 보이지 않는 보기를 변경된것으로 표시한다. 이것은 사용자가 보기를 변경할 때 절환하는 보기를 갱신한다는것을 담보한다. 단순히 2개 보기를 변경된것으로 표식하고 둘다 갱신하지만 천천히 갱신하는것이 더 효과있다. 결국 사용자는 하나의 보기만 사용하므로 다른 보기를 필요없이 갱신하는데 시간을 랑비하지 않게 한다.

QFile과 QRegExp를 사용하므로 관련머리부를 포함한다. (Includes(in Implementation)를 오른쪽 단추로 찰칵하고 New를 선택한다. qfile.h라고 입력하고 Enter를 누른다. 이 과정을 반복하여 qregexp.h를 추가한다.)

실현파일의 선두에 다음의 선언들을 추가한다.

"qfile.h"

"qregexp.h"

정규표현

우리가 사용하는 정규표현은 다음의 부분들로 나눌수 있다.

정규표현:	^	\\s*	(\\d+)	\\s+	(\\d+)	\\s+	(\\d+)	\\s+	(\\S+.*)	$
부분:	A	B	C	D	C	D	C	D	E	F
괄호:			cap(1)		cap(2)		cap(3)		cap(4)	

부분A는 정규표현이 문자렬의 시작과 일치해야 한다는것을 말하며 부분F는 정규표현이 문자렬끝과 일치해야 한다는것을 말한다. 따라서 정규표현은 문자렬과 완전히 일치하거나 일치하지 말아야 한다. B부분은 0이상의 공백(즉 유효공백)과 일치하고 부분D는 1개이상의 공백(즉 매개 수자사이의 짬)과 일치한다. 부분C는 한개이상의 수자들 즉 수와 일치한다. 부분E는 다른것이 뒤따르는 한개이상의 비공백 즉 색이름과 일치한다.

괄호는 그것들이 둘러싸는 일치부분들을 둘러싸는데 쓰인다. 괄호에 넣은 부분은 1로부터 계수된다.

- fileSaveAs()

```
void MainForm::fileSaveAs()
{
    QString filename = QFileDialog::getSaveFileName( QString::null,
          "Colors (*.txt)", this, "file save as", "Color Tool -- File Save As" );
    if ( ! filename.isEmpty() ) {
        int ans = 0;
        if ( QFile::exists( filename ) )
            ans = QMessageBox::warning( this, "Color Tool -- Overwrite File",
                        QString( "Overwrite\n'%1'?" ) .arg( filename ),
                        "&Yes", "&No", QString::null, 1, 1 );
        if ( ans == 0 ) {
            m_filename = filename;
            fileSave();
            return;
        }
    }
    statusBar()->message( "Saving abandoned", 2000 );
}
```

사용자가 편집하고 보관하지 않은 자료를 보관하려고 시도하거나 현존자료를 새 이름으로 보관하려고 한다면 이 처리부가 호출된다. 사용자에게 파일이름을 선택할수 있는 표준파일대화칸이 제시된다. 이미 파일이름이 존재하면 계속(덧쓰기)하겠는가 또는 취소하겠는가 하는 선택이 주어진다. 파일이름이 존재하지 않거나 혹은 사용자가 계속하기로 선택하였다면 m_filename성원이 설정되고 fileSave()가 호출된다.

- fileSave()

```
void MainForm::fileSave()
{
    if ( m_filename.isEmpty() ) {
        fileSaveAs();
        return;
    }
```

```
    QFile file( m_filename );
    if ( file.open( IO_WriteOnly ) ) {
        QTextStream stream( &file );
        if ( ! m_comments.isEmpty() )
            stream << m_comments.join( "\n" ) << "\n";
        QMap<QString,QColor>::ConstIterator it;
        for ( it = m_colors.constBegin(); it != m_colors.constEnd(); ++it ) {
            QColor color = it.data();
            stream << QString( "%1 %2 %3\t\t%4" ).
                        arg( color.red(), 3 ).
                        arg( color.green(), 3 ).
                        arg( color.blue(), 3 ).
                        arg( it.key() ) << "\n";
        }
        file.close();
        setCaption( QString( "Color Tool -- %1" ).arg( m_filename ) );
        statusBar()->message( QString( "Saved %1 colors to '%2'" ).
                            arg( m_colors.count() ).
                            arg( m_filename ), 3000 );
        m_changed = FALSE;
    }
    else
        statusBar()->message( QString( "Failed to save '%1'" ).
                            arg( m_filename ), 3000 );

  }
```

현재파일이름이 없으면 fileSaveAs()를 호출하고 사용자가 파일이름을 제공하면 그 함수는 이것을 호출한다.

우선 설명행을 쓴다. 이것은 적재하고 보관한 파일이 같지 않을수 있다는것을 의미한다. (실례로 원시파일이 도처에 분산되여있는 설명문들을 가지고있다면 우리가 보관한 판이 선두에 모든 설명을 가지게 된다.) 그다음 m_colors매프의 매개 색을 순환하면서 rgb.txt파일형식으로 써넣는다.

- fileExit()

```
  void MainForm::fileExit()
```

```
{
   if ( okToClear() ) {
      QApplication::exit( 0 );
   }
}
```

이것은 이 함수의 두번째 갱신이다. 이제는 사용자가 보관하지 못한 변경을 보관할 기회를 가지게 되였다. (후에 이 함수의 제3판과 최종판을 만든다. 이때 사용자설정의 보관을 론한다.)

프로그람을 구축하고 실행한다. 체계에 rgb.txt가 있다면 그것을 적재해보고 시험을 목적으로 새로운 이름으로 보관해본다. 이런 파일이 없다면 표준색을 보관하고 그것을 리용해본다. 다음에 자기의 색파일들을 생성할수 있도록 색의 추가 및 삭제방법을 설명한다.

10) 편집선택코드의 편집

새로운 색의 추가, 탐색, 사용자선택의 조종은 모두 전용대화칸을 요구하므로 그것들은 대화칸을 론하는 3절에로 넘기기로 한다.

- editCut()

```
void MainForm::editCut()
{
   QString name;
   QWidget *visible = colorWidgetStack->visibleWidget();
   statusBar()->message( QString( "Deleting '%1'" ).arg( name ) );

   if ( visible == tablePage && colorTable->numRows() ) {
      int row = colorTable->currentRow();
      name = colorTable->text( row, 0 );
      colorTable->removeRow( colorTable->currentRow() );
      if ( row < colorTable->numRows() )
         colorTable->setCurrentCell( row, 0 );
      else if ( colorTable->numRows() )
         colorTable->setCurrentCell( colorTable->numRows() - 1, 0 );
      m_icons_dirty = TRUE;
   }
   else if ( visible == iconsPage && colorIconView->currentItem() ) {
      QIconViewItem *item = colorIconView->currentItem();
      name = item->text();
```

```
        if ( colorIconView->count() == 1 )
            colorIconView->clear();
        else {
            QIconViewItem *current = item->nextItem();
            if ( ! current )
                current = item->prevItem();
            delete item;
            if ( current )
                colorIconView->setCurrentItem( current );
            colorIconView->arrangeItemsInGrid();
        }
        m_table_dirty = TRUE;
    }

    if ( ! name.isNull() ) {
        m_colors.remove( name );
        m_changed = TRUE;
        statusBar()->message( QString( "Deleted '%1'" ).arg( name ), 5000 );
    }
    else
        statusBar()->message(QString("Failed to delete '%1'").arg(name), 5000);
}
```

사용자가 표보기를 보고있으면 현재 행을 삭제한다. 새로운 현재 세포를 삭제된 렬뒤의 것으로 설정하거나 삭제한것이 마지막이면 그앞의것으로 설정한다. 다른 보기(그림기호보기)를 변경된것으로 표식한다. 사용자가 보기들을 절환하면 그것이 갱신된다는것을 담보하기 위해서이다. 마찬가지로 사용자가 그림기호보기를 보고있으면 다음(없으면 이전)항목을 현재 항목으로 만들고 항목들이 있던 보기를 삭제한다. 그다음 표보기를 변경된것으로 표식한다. 색을 삭제하였다면(즉 보기들중 하나에 현재색이 있었다면) m_colors매프에서 그것을 삭제하며 자료를 변경된것으로 표식한다.

- editCopy()

```
void MainForm::editCopy()
{
    QString text;
    QWidget *visible = colorWidgetStack->visibleWidget();
```

```
    if ( visible == tablePage && colorTable->numRows() ) {
        int row = colorTable->currentRow();
        text = colorTable->text( row, 0 );
    }
    else if ( visible == iconsPage && colorIconView->currentItem() ) {
        QIconViewItem *item = colorIconView->currentItem();
        text = item->text();
    }
    if ( ! text.isNull() ) {
        QColor color = m_colors[text];
        switch ( m_clip_as ) {
            case CLIP_AS_HEX: text = color.name(); break;
            case CLIP_AS_NAME: break;
            case CLIP_AS_RGB:
                    text = QString( "%1,%2,%3" ).
                        arg( color.red() ).
                        arg( color.green() ).
                        arg( color.blue() );
                    break;
        }
        clipboard->setText( text );
        statusBar()->message( "Copied '" + text + "' to the clipboard" );
    }
}
```

이 함수에서 현재표의 행으로부터 색이름(또는 보기에 따라 현재 그림기호)을 얻는다. 그다음 QString을 오려둠판에 복사하려고 하는 본문으로 설정하고 그것을 복사한다.

이 절에서는 표준기본창문형식의 응용프로그람을 작성하였다. 그리고 차림표와 도구띠, 기본창문부품(QWidgetStack)들을 실현하였다. 또한 신호-처리부련결을 만들고 많은 전용처리부들을 실현하였다.

제3절. 대화칸작성

이 절에서는 colortool응용프로그람을 완성하는데 필요한 대화칸들을 만든다. 앞절에서 생성한 모든 대화칸은 기본창문으로부터 호출된다. 모달 및 이행허용대화칸을 생성하고 사용하는 방법과 Qt Designer에서 Qt의 배치클라스들을 사용하여 잘 비레되는 폼들을 생성하는 방법을 학습한다.

1. 색추가

Qt는 이미 색선택대화칸을 펼치는 정적함수를 가지고있다. 그러나 색을 선택할 뿐아니라 거기에 이름도 주어야 한다. 실례로 사용자에게 색선택대화칸을 제공하고 색을 선택하면 그 이름을 묻는다.



그림 1-46. 색선택대화칸

1) 련결의 만들기

기본폼을 생성하였을 때 editAddAction이라는 작용을 만들었다. 차림표띠(Edit차림표)에 이 작용을 추가하고 도구띠에도 추가하였다. 이제는 이 작용을 처리부에 련결하여 색을 추가하게 할수 있다.

Edit|Connections을 눌러서 View and Edit Connections대화칸을 열고 New를 선택하여 새 련결을 생성하며 Sender를 editAddAction로, 신호를 activated()로, receiver를 MainForm으로 변경한다. 련결하려는 새 처리부를 창조해야 한다. Edit Slots를 선택하여 Edit Functions대화칸을 열고 New Function를 찰칵하고 처리부의 이름을 editAdd()로 변경한 다음 OK를 찰칵한다. 이제 련결하는 처리부를 editAdd()로 변경한 다음 OK를 찰칵하여 대화칸을 닫는다.

- 창문부품의 선택

개별적창문부품을 선택하기 위하여 창문부품자체를 누르거나 Object Explorer에서 Name을 찰칵한다. 그룹을 선택하려면 붉은색 륜곽선밖의 부분을 누르거나 Object Explorer에서 Name을 찰칵한다.

배치안에 있는 창문부품들사이의 짬에 창문부품을 삽입하려고 한다면 새 창문부품용의 도구띠단추를 누르고 짬안에서 마우스를 찰칵한다. Qt Designer는 배치를 파괴하려고 하는가를 묻는데 여기서 Break Layout를 누르면 배치는 파괴되고 창문부품이 삽입된다. 그다음 배치 및 다시 배치하려는 창문부품과 그룹들을 선택할수 있다. 그룹을 선택하고 Break Layout도구띠단추나 Ctrl+B를 누름으로써 같은 효과를 얻을수 있다.

5가지의 각이한 방법으로 여러개의 창문부품들을 선택할수 있다.

① 첫 창문부품을 선택하고 다른 창문부품들을 Shift를 누른 상태에서 마우스로 찰칵한다.

② 첫 창문부품을 Ctrl을 누른 상태에서 마우스로 누른 다음 다른 창문부품들을 Ctrl을 누르면서 마우스로 찰칵한다. 이것은 첫째 수법과 비슷하지만 다른 창문부품(실례로 그룹)안에 있는 창문부품들을 선택하게 한다.

③ 폼을 선택하고 선택창을 끌면서 포함하려는 모든 창문부품들에 접근한다.

④ 첫 창문부품을 Ctrl를 누른 상태에서 마우스로 선택하고 선택창을 끌면서 다른 창문부품들에 접근한다. 이것은 다른 창문부품(례하면 그룹)안에 있는 창문부품들을 선택하게 하는 이전의 수법들과 다르다.

⑤ Object Explorer의 Objects타브를 찰칵한다. Object Explorer에서 첫 창문부품을 선택한 다음 다른 창문부품을 Shift를 누른 상태에서 마우스로 찰칵한다. 이것은 복잡한 배치가 많이 있을 때 창문부품들을 선택하는데 특히 효과있다.

여러개의 창문부품들이 선택될 때 그 공통속성들이 Property Editor에 표시된다. Property Editor에서 이루어진 변경은 선택된 모든 창문부품들에 적용된다. 이것은 공통적인 최소최대크기, 색, 크기조절방법, 유표, 서체 등을 설정하는데 특히 효과있다.

2) 대화칸작성

- 배치

배치(layout)는 창문부품, 창문부품그룹, 배치들을 수평, 수직 및 살창으로 배치하는 도구를 제공한다. 배치를 폼에 사용하면 그것이 포함하는 창문부품들은 사용자가 창문크기를 조절하는데 따라 자동적으로 비례하여 조절된다. 이것은 비례를 얻기 위한 코드를 쓰지 않으므로 절대크기와 위치를 사용하는것보다 더 좋으며 사용자들이 무릎형콤퓨터를 가지는가 대형화면 탁상콤퓨터를 가지고있는가에 관계없이 자기 화면의 대부분을 차지할수 있다. 배치는 여백과 창문부품간격의 표준크기를 사용하여 응용프로그람들에 일관적이고 비례적인 외관을 주게 한다. 또한 배치는 절대위치지정보다 사용하기 쉽고 빠르다. 근사위치의 폼으로 창문부품들을 배치할수 있으며 창문부품들의 크기와 비례를 정확히 설정하는데 배치도구를 사용한다.

- 창문부품의 추가

Qt의 정적대화칸들중 하나를 사용하여 사용자가 색을 선택하게 할수 있으나 우리는 자

체의 대화칸을 이름을 지정하여 만들어 색을 얻게 한다. 이제 그 대화칸을 만들어보자.

File|New를 눌러서 New File대화칸을 열고 Dialog를 누른 다음 OK를 찰칵한다. 새 폼의 모서리를 끌기하여 크기를 더 작게 만든다. Property Editor에서 폼이름을 ColorNameForm로, 제목을 Color Tool -- Color Name으로 변경한다. File|Save를 누르고 Save를 선택하여 그것을 보관한다.

그다음 대화칸에 창문부품들을 추가한다. 위치와 크기를 정확히 지정하지 않아도 된다. Qt Designer는 폼을 간단히 배치할수 있게 한다.

사용자가 선택한 색을 표시하는 QLabel을 만들어야 한다. Toolbox에서 TextLabel도구를 누르고 폼의 왼쪽을 선택한다. 표식의 이름속성을 colorLabel로 변경하고 본문을 text속성에서 삭제한다. pixmap속성의 생략단추를 누르고 editraise.png 화상을 선택한다. minimumSize속성의 width보조속성을 80으로, scaledContents속성을 True로 변경한다.

TextLabel도구를 다시 선택한 다음 colorLabel의 오른쪽, 폼의 웃부분을 선택하고 text속성을 &Name으로 변경한다. &가 표시된다. 이것은 QLabel이 초점을 받아들일수 없으며 그것은 아직 초점창문부품(buddy)을 지정하지 않았기때문이다.

LineEdit도구를 선택하고 Name표식자의 오른쪽을 선택한 다음 다시 폼의 웃부분을 선택한다. name속성을 colorLineEdit로 변경한다.

Name표식자를 선택하고 buddy속성을 colorLineEdit로 변경한다. 지금 &는 표시되지 않고 Alt+N은 colorLineEdit에 초점을 설정한다.

PushButton도구를 선택하고 colorLabel아래를 찰칵한다. 단추의 이름속성을 okPushButton로, text속성을 OK로, default속성을 True로 변경한다.

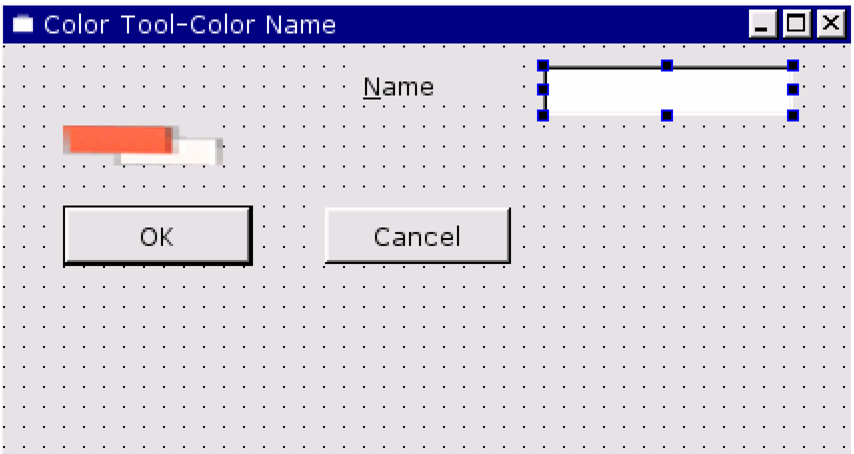


그림 1-47. 색선택대화칸에 창문부품들의 추가

PushButton단추를 다시 선택하고 OK단추의 오른쪽을 찰칵한다. 이 단추의 name속성을 cancelPushButton으로, text속성을 Cancel로 변경한다.

- 창문부품의 배치

이제는 필요한 창문부품들을 만들어 대체로 배치하였으므로 그것들을 정확히 배치할 준비가 되였다.

창문부품배치과정은 기본적으로 2단계이다.

2개이상의 창문부품(또는 배치)을 선택한다.

배치방법(수직, 수평, 살창)을 선택항목들에 적용한다.

Qt Designer는 창문부품과 배치를 선택하는 몇가지 다른 방법을 제공한다. 그 사용에서 문제될것은 없으며 일부는 일정한 상황에서 다른것보다 더 편리하다. 폼을 배치할 때 창문부품들을 선택하는 몇가지 방법들을 보여준다.

Name표식자과 행편집칸을 나란히 배치한다. 그다음 단추들도 같은 방법으로 배치하고 끝으로 colorLabel과 련관된 2개를 배치한다.

폼을 선택하여 창문부품선택을 해제한다. Name표식자우의 폼을 선택하고 신축할수 있는 검은색 장방형을 끌기하여 Name표식자과 행편집칸에 모두 닿도록 하고 마우스를 놓는다.(마우스를 놓을 때 그것들에 닿아야 한다.) 그러면 Name표식자과 행편집칸이 선택된다. Lay Out Horizontally도구띠단추를 찰칵한다. 배치에 포함된 항목들이 붉은색 장방형안에 나타난다. (이것은 오직 Qt Designer에서 시각적인 암시로서 표시되며 미리보기방식이나 실행하는 폼에서는 나타나지 않는다.)

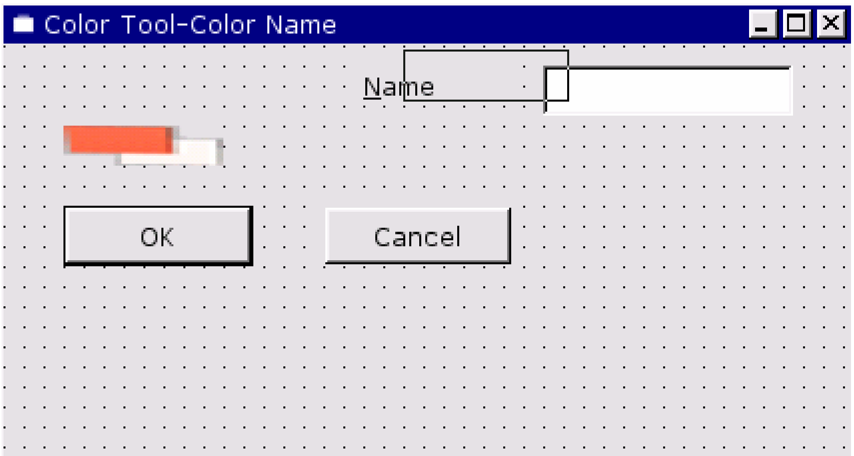


그림 1-48. Name표식자과 행편집칸의 수평배치

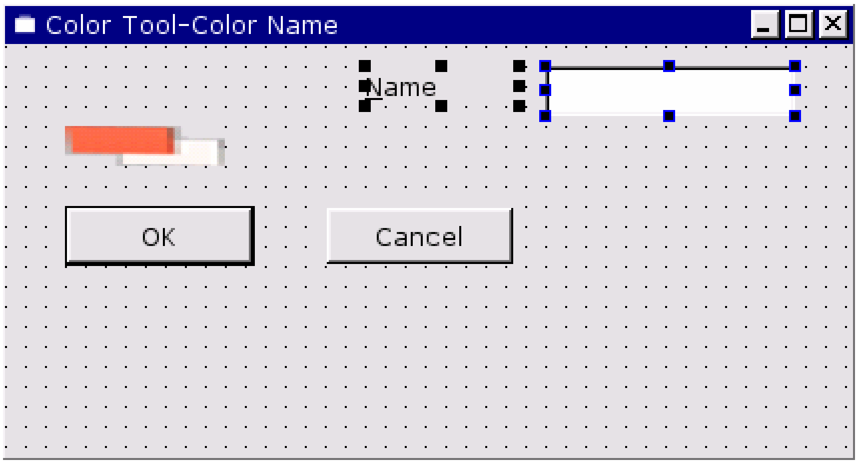


그림 1-49. 수평배치후 Name표식자과 행편집칸

폼을 선택하여 창문부품선택을 해제한다. Object Explorer의 Objects타브에서 cancelPushButton을 선택한다. 이제 Objects타브에서 Shift를 누르면서 OK단추를 누르고(Objects타브는 오직 한개 객체만 강조표시하지만 폼은 두개 단추가 선택된다는 것을 보여준다.) Lay Out Horizontally도구띠단추를 찰칵한다.

폼을 선택하여 창문부품선택을 해제한다. Objects Explorer에서 Layout1(Name표식자과 행편집이 있다.)를 누르고 Shift를 누르면서 Layout2(단추들이 있다.)를 마우스로 찰칵하여 폼에서 두개 배치를 선택하고 Lay Out Vertically도구띠단추를 찰칵한다.

이제는 2개의 배치와 colorLabel창문부품을 얻었으며 폼과 관련하여 그것들을 배치한다.

폼을 선택하여 창문부품의 선택을 해제한다. Lay Out Horizontally도구띠단추를 찰칵하여 폼을 배치한다.

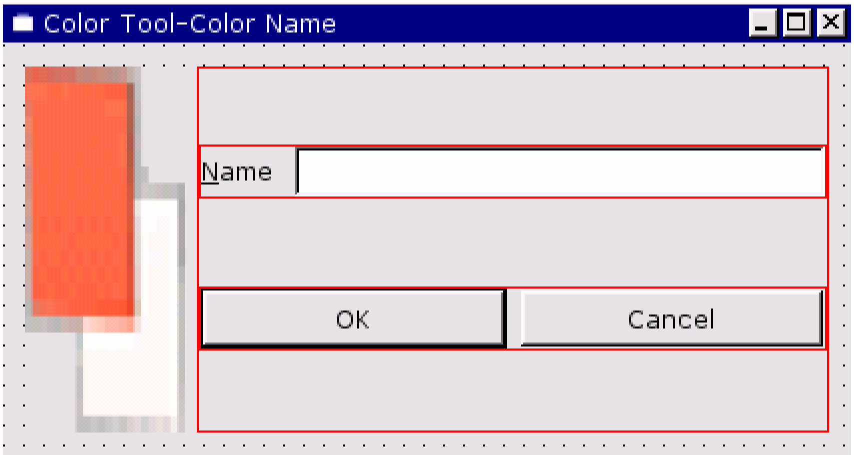


그림 1-50. 폼의 수평배치

폼을 미리보기(Ctrl+T)하고 그 크기를 조절한다. 폼을 더 크게 하면 단추들은 너무 커져서 볼품없이 된다. 더우기 단추들이 폼의 바닥에 놓이지 않는다. 문제는 폼을 늘굴 때 사용하지 않는 공간이 많으므로 단추들이 이 공간을 사용하지 않도록 하는것이다. 이것은 수축자를 삽입하여 달성한다.

수축자들을 삽입하고 폼을 다시 배치하도록 하기 위하여 폼배치를 파괴하여야 한다. Layout3의 붉은색장방형을 선택하고 전체폼의 배치를 선택하고 (Object Explorer에서 배치를 선택하여 간단히 얻을수 있다.) Break Layout도구띠단추를 찰칵한다.

단추들의 왼끝에 수축자를 하나 추가하고 그것을 단추들과 배치할수 있으나 여유배치창조를 보관하기 위하여 그대신 단추들을 포함하는 배치를 파괴하고 수축자를 가진 단일한 배치로서 그것들을 배치한다. 단추들중 하나를 선택하고 Break Layout(또는 Ctrl+B)를 누른다. OK단추를 약 절반폭으로 그 왼쪽에 짬을 주어서 대충 크기를 조절한다. Toolbox안에서 또는 도구띠우에서 Spacer도구를 누르고 OK단추왼쪽의 폼을 선택하고 수평으로 끌기한다. 지금 수평수축자(푸른색 용수로 가리킨다.)는 OK단추의 왼쪽에 있다.

수축자를 이미 선택하였으므로 단추들을 포함하도록 선택을 간단히 확장한다. Shift를 누르면서 OK를 찰칵하고 취소단추를 찰칵하여 2개 단추와 수축자를 선택하고 Lay Out Horizontally(또는 Ctrl+H)를 누른다. 지금 수축자는 자기 위치에 있고 여유공간을 소비할수 있다.

이제는 행편집칸을 포함하는 배치와 방금 생성한 배치(단추들을 포함한다.)사이에 수축자를 넣어서 그것들사이의 여유공간을 없애버릴수 있다. Spacer도구를 선택하고 행편집과 단추들사이의 폼을 누르고 수직으로 끌기한다. 수축자는 이미 선택되였지만 선택을 확장하여 배치를 둘다 포함해야 한다. Object Explorer의 Objects타브에서 Shift건을 누르면서 배치들을 찰칵하고 Lay Out Vertically(또는 Ctrl+L)을 누른다.

그러면 폼자체를 배치하자. 폼을 선택하여 창문부품이나 배치들의 선택을 해제한다. 그다음 Lay Out Horizontally를 찰칵한다.

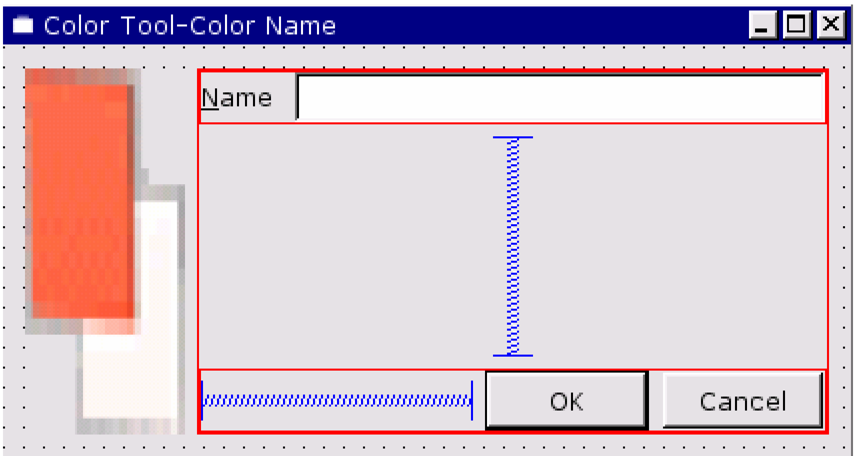


그림 1-51. 수축자들의 추가

지금까지는 폼이 이전과 크게 차이나지 않았다. 폼을 미리보기(Ctrl+T)하고 크기를 조절한다. 폼을 작게 하든 크게 하든 문제는 없고 언제나 잘 비례되여있다.

이것이 바로 고정크기보다 배치를 리용하는것의 우점이다. 프로그람이 각이한 언어로 번역된다면 어떤 코드도 작성함이 없이 표식자의 크기를 자동적으로 비례되게 조절하므로 특별히 효과있다. 그리고 사용자들은 거대한 탁상화면으로부터 작은 무릎형화면에 이르기까지의 넓은 범위의 화면크기에 사용할수 있다. (필요하다면 고정크기와 위치를 사용할수 있다.)

Qt Designer에서 폼크기를 조절한다. 폼은 최소크기를 가진다. 이것은 폼이 포함하는 모든 창문부품들이 최소크기를 가지기때문이다. (필요하다면 최소크기를 재정의할수 있다.) Qt Designer에서 폼을 만드는 크기는 폼의 기정크기이다.

폼을 미리보기할 때 타브순서가 정확한 경우에 Tab를 눌러서 초점을 받아들이는 창문부품들사이를 이동할수 있다. 타브순서가 정확하지 않으면 간단히 변경할수 있다.

- 타브순서의 변경

건반사용자들은 Tab건을 눌러서 폼우의 창문부품들사이로 초점을 이동한다. 초점이 이동하는 순서를 타브순서라고 한다.

폼의 타브순서를 변경하려면 Tab Order도구띠단추를 찰칵한다. 이것은 Qt Designer를 타브순서방식에 넣는다. 푸른색원안의 수값이 초점을 받아들일수 있는 매개 창문부품열에 나타난다.

타브순서를 변경하려면 초점을 받아들이는 순서로 각 창문부품들을 선택한다. 모든 창문부품들이 정확한 타브순서를 가지면 중지한다. Esc를 눌러서 타브순서방식을 끝낸다. 미리보기(Ctrl+T)하고 Tab건을 누름으로써 타브순서를 시험할수 있다.

이전의 타브순서가 요구되면 Edit|Undo(또는 Ctrl+Z)를 누른다.

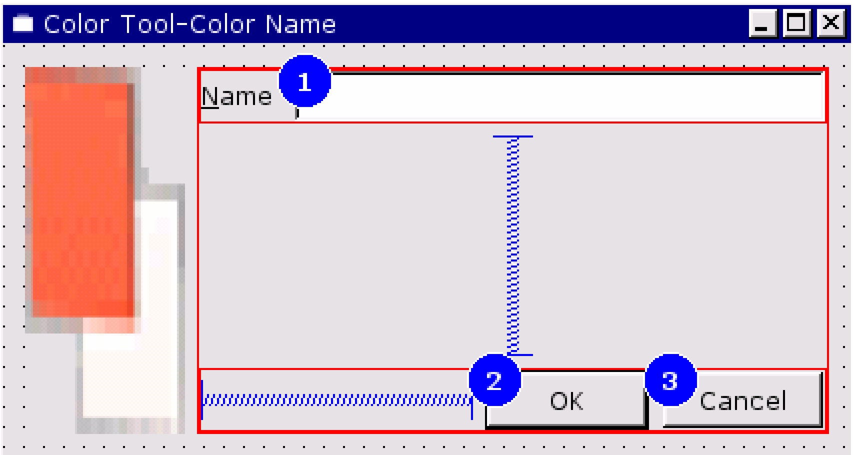


그림 1-52. 타브순서설정

- 창문부품의 련결

2개의 단추 즉 OK단추와 Cancel단추를 조종해야 한다. 사용자가 OK를 찰칵하면 색이름이 비여있지 않는 경우와 이미 사용중에 있지 않는 경우 색이름만 받아들인다. (rgb.txt형식은 색의 중복을 허용하지만 그것들을 추가하지 않도록 선택한다.) 사용자가 Cancel을 누르면 대화칸을 닫는다.

Ctrl+T를 눌러서 폼을 미리보기한다. Cancel단추를 누르면 아무일도 하지 않는다.

제일 간단한 Cancel단추부터 련결한다. Edit|Connections를 눌러서 View and Edit Connections대화칸을 열고 New를 눌러서 새 련결을 생성한다. Sender를 cancelPushButton로, Signal를 clicked()로, Receiver를 ColorNameForm로, 처리부를 reject()로 설정한다. 이 기능은 미리 정의된 신호와 미리 정의된 처리부를 사용하여 신호-처리부련결을 통하여 얻어지므로 미리보기방식에서 작업한다. OK를 찰칵하여 대화칸을 닫은 다음 Ctrl+T를 눌러 미리보기한다. Cancel단추를 찰칵하면 미리보기방식에서도 폼이 닫겨진다.

이제는 OK단추를 련결한다. Edit|Connections를 눌러서 View and Edit Connections대화칸을 열고 New를 눌러서 새 련결을 생성한다. Sender를 okPushButton로, Signal를 clicked(), Receiver를 ColorNameForm로 설정한다. 사용자의 입력을 확증할수 있도록 자체의 전용처리부를 호출하려고 한다. Edit Slots를 눌러서 Edit Functions대화칸을 펼친다. New Function을 선택하고 함수이름을 validate()로 변경하고 OK를 찰칵한다. 수신기의 처리부를 새로 생성한 validate()처리부로 설정한다. OK를 찰칵하여 대화칸을 닫는다.

- 대화칸코드작성

validate()전용처리부의 코드를 작성해야 한다. 이 처리부는 입력한 색이름이 이미 존재하는가 확인하기 위하여 검사하므로 색이름들의 폼대역목록을 설정할수 있는 함수를 제공해야 한다.

Project Overview창문에서 colornameform.ui.h를 누르고 코드편집기를 호출한다. 편집기는 하나의 빈 처리부 validate()를 보여준다.

머리부파일들과 함께 색이름들을 보관하는 폼대역변수를 추가해야 한다. 기본폼에 대하여 수행한것과 같은 방법으로 Objects Explorer의 Members목록에서 적당한 부분을 오른쪽 단추로 찰칵하고 이 정보를 추가할수 있다. 그대신에 다른 수법으로 폼에 직접 필요한것을 추가한다. 필요하다면 두 수법을 리용할수 있다.

- 성원추가와 .ui.h코드작성

Object Explorer의 Members타브에서 Includes (in Implementation)에 머리부를 추가하는 수법과 .ui.h파일에 직접 머리부를 입력하는 수법사이에 차이가 없다.

상황은 변수들에 따라 다르다. 이것들을 Members, Class Variables에 추가하면

클라스정의에 비공개변수로서 포함된다. .ui.h파일의 짝대기에 그것들을 입력하면 폼대역변수로 된다.

다음과 같이 머리부를 추가한다.

```
#include <qcolor.h>
#include <qmap.h>
#include <qstring.h>
```

validate()함수우에 우의 행들을 입력한다.

또한 색목록을 보관하는 변수가 요구된다.

```
QMap<QString,QColor> m_colors;
```

이 행을 추가하고 국부 m_colors매프에 색들을 보관한다.

또한 호출자가 현재색을 가진 m_colors매프를 구성하기 위하여 호출할수 있는 함수를 요구한다.

```
void ColorNameForm::setColors( const QMap<QString,QColor>& colors )
{
   m_colors = colors;
}
```

이제는 색이름목록을 얻는 수단이 있으므로 validate()함수를 작성해보자.

```
void ColorNameForm::validate()
{
   QString name = colorLineEdit->text();
   if (!name.isEmpty() && (m_colors.isEmpty() || ! m_colors.contains( name )) )
      accept();
   else
      colorLineEdit->selectAll();
}
```

함수는 사용자가 입력한 본문을 시험한다. 무엇인가 입력하였는데 그것이 색목록에 존재하지 않으면 accept()를 호출한다. 이것은 폼을 닫고 호출측에 true값을 돌려준다. (reject()는 사용자가 Cancel을 누르면 호출되며 false값을 돌려준다.) 이미 색이 있으면 단지 그것을 선택하거나 통보창문을 펼칠수 있다.

이제는 대화칸이 완성되였다. 다음 단계는 editAdd()처리부의 코드를 작성하여 기본폼으로부터 그것을 사용하게 하는것이다.

3) 대화칸리용

ColorNameForm대화칸은 기본폼으로부터 호출된다. 호출자는 우선 Qt의 색선택정적대화칸을 호출하고 사용자가 색을 선택하면 ColorNameForm전용대화칸을 호출한

다. 색선택대화칸을 사용하려고 하므로 적당한 머리부파일을 요구한다. 또한 대화칸의 colorLabel(그것을 선택된 색으로 설정하기 위하여)과 대화칸의 행편집(색이름을 얻기 위하여)을 호출하므로 이것들에 해당한 머리부들도 요구한다.

Project Overview창문에서 MainForm을 찰칵하면 Object Explorer는 기본폼의 객체들을 표시한다.

Object Explorer의 Members타브를 선택한다. Includes (in Implementation)를 오른쪽 단추로 선택하고 Edit를 눌러서 Edit Includes(in Implementation)대화칸을 연다. Add를 누른 다음 qcolordialog.h라고 입력한다. 다시 Add를 선택하고 qlabel.h라고 입력한다. 마찬가지로 qlineedit.h를 추가한다. 또한 방금 생성한 폼용의 머리부를 포함해야 하므로 colornameform.h를 추가하고 Enter를 누른다. 다음 Close를 찰칵한다.

이제는 머리부(실현에서)들에 다음의 선언들을 추가한다.

·"qcolordialog.h"

·"qlabel.h"

·"qlineedit.h"

·"colornameform.h"

editAdd()처리부의 코드를 입력할 준비가 되였다. Project Overview에서 mainform.ui.h를 선택하여 코드편집기를 호출한다.

- editAdd()

```
void MainForm::editAdd()
{
    QColor color = white;
    if ( ! m_colors.isEmpty() ) {
        QWidget *visible = colorWidgetStack->visibleWidget();
        if ( visible == tablePage )
            color=colorTable->text(colorTable->currentRow(),
                                            colorTable->currentColumn());
        else
            color = colorIconView->currentItem()->text();
    }
    color = QColorDialog::getColor( color, this );
    if ( color.isValid() ) {
        QPixmap pixmap( 80, 10 );
        pixmap.fill( color );
```

```
            ColorNameForm *colorForm = new ColorNameForm(this,"color",TRUE);
            colorForm->setColors( m_colors );
            colorForm->colorLabel->setPixmap( pixmap );
            if ( colorForm->exec() ) {
               QString name = colorForm->colorLineEdit->text();
               m_colors[name] = color;
               QPixmap pixmap( 22, 22 );
               pixmap.fill( color );
               int row = colorTable->currentRow();
               colorTable->insertRows( row, 1 );
               colorTable->setText( row, COL_NAME, name );
               colorTable->setPixmap( row, COL_NAME, pixmap );
               colorTable->setText( row, COL_HEX, color.name().upper() );
               if ( m_show_web ) {
                  QCheckTableItem *item = new QCheckTableItem( colorTable, "" );
                  item->setChecked( isWebColor( color ) );
                  colorTable->setItem( row, COL_WEB, item );
               }
               colorTable->setCurrentCell( row, 0 );
               (void) new QIconViewItem( colorIconView, name, colorSwatch( color ) );
               m_changed = TRUE;
            }
         }
      }
```

이 함수의 코드는 아주 길지만 리해하기 힘들지 않다. 기정색을 흰색으로 설정하는 것으로 시작한다. m_colors매프에 색들이 있으면 기정색을 현재보기에 보여주는 현재색으로 설정한다. 그다음 Qt의 정적getColor()대화칸을 호출하고 거기에 기정색을 넘긴다.(사용자가 취소하면 무효색이 돌아온다.)

사용자가 색을 선택하면 전용대화칸에 선택한 색을 보여주려고 하므로 픽스매프를 생성하고 그것을 선택된 색으로 채운다. ColorNameForm의 실례를 이행금지대화칸(셋째인수는 TRUE)로 생성한다. 그다음 setColors()함수를 호출하여 (validate()함수가 정확히 작업하도록) m_colors매프에 색들을 설정한다. colorLabel의 픽스매프를 방금 창조한 픽스매프 즉 사용자의 선택색으로 된 장방형으로 설정한다.

대화칸을 실행(exec())한다. 사용자가 OK를 찰칵하면(그리고 입력한 색이름이 유효

이면) 호출은 true값을 돌려준다. 이 경우에 행편집으로부터 입력한 이름을 얻어서 사용자가 준 이름과 선택한 색을 리용하여 m_colors매프에 새 항목을 창조한다.

이 시점에서 보기들을 간단히 변경된것으로서 표식하고 repopulate()을 호출할 대신 매개 보기에 직접 새 색을 추가함으로써 완전갱신으로 인한 추가비를 절약한다. (수천개의 색이 있으면 고려할만하다.)

픽스매프를 창조하고 그것을 새로운 색으로 채운 다음 populate()함수에서와 같은 방법으로 표에 새로운 행을 삽입하고 새로운 색값으로 렬을 설정한다. 마찬가지로 그림기호보기용의 새로운 그림기호를 창조한다. 끝으로 자료를 변경된것으로 표식하여 사용자가 이것을 보관하지 않고 완료하거나 다른 색파일을 적재하려고 하면 보관하겠는가를 묻게 한다.

응용프로그람의 건설, 실행한다. 이제는 자체의 색을 추가할수 있다.

응용프로그람은 기본적으로 완성되였다. 색파일을 적재하고 보관할수 있고 그것들을 표보기나 그림기호보기에 표시할수 있으며 사용자는 색들을 추가삭제할수 있다. 그러나 사용자가 수백수천개의 색을 가지고있으면 특정색을 찾는데 품이 든다. 또한 사용자가 응용프로그람을 실행할 때는 항상 기정창문크기, 보기, 기타 설정으로 기동한다. 사용자가 응용프로그람을 어떻게 완료하였는가를 기억하고 그 선택을 재생하면 훨씬 더 좋다. 다음의 2개 소절에서 비모달탐색대화칸와 모달선택대화칸의 창조를 통하여 이 문제를 고찰한다.

2. 색탐색

이 선택수법은 색추가와 비슷하다. 대화칸을 만들고 련결을 만들고 코드를 쓰고 그것을 호출하는 처리부의 코드를 작성한다. (이미 기본폼에서 설정된 련결을 가지고있는데 그것은 기본창문위자드에 의하여 자동적으로 창조되였다.)



그림 1-53. 색탐색대화칸

1) 대화칸작성

File|New를 선택하여 New File대화칸을 열고 Dialog를 선택하고 OK를 찰칵한다. 폼의 모서리를 끌어서 폼을 작게 만든다. Property Editor를 사용하여 폼의 name속성을 FindForm로, caption속성을 "Color Tool -- Find Color"로 변경한다.

폼을 보관하기 위하여 Ctrl+S(혹은 File|Save)를 선택하고 Save를 찰칵한다.

- 창문부품의 추가

Toolbox에서 TextLabel도구를 선택하고 폼의 왼쪽을 선택한다. text속성을

"&Look for"로 변경한다. (&는 Alt+L지름건을 위한 초점창문부품 buddy를 아직 제공하지 않았으므로 폼에 표시된다.)

LineEdit도구를 선택하고 폼우에서 Look for표식자의 오른쪽을 선택하고 name속성을 findLineEdit로 변경한다. Look for표식자를 누르고 buddy속성을 findLineEdit로 변경한다.

PushButton도구를 선택하고 폼에서 행편집아래를 선택한다. name속성을 findPushButton로, text속성을 &Find로, default속성을 True로 변경한다.

PushButton도구를 다시 선택하고 폼우에서 Find단추의 오른쪽을 찰칵한다. name속성을 closePushButton로, text속성을 Close로 변경한다.

단추들우에 그리고 단추들의 왼쪽에 수축자를 요구하는 이전 폼을 가지고 실험하여본다. Spacer도구를 선택하고 Find단추의 왼쪽을 선택하고 좀 오른쪽으로 수평으로 끌고가서 놓는다. Spacer도구를 다시 선택하고 Find단추우를 누르고 수직으로 좀 끌고가서 놓는다.

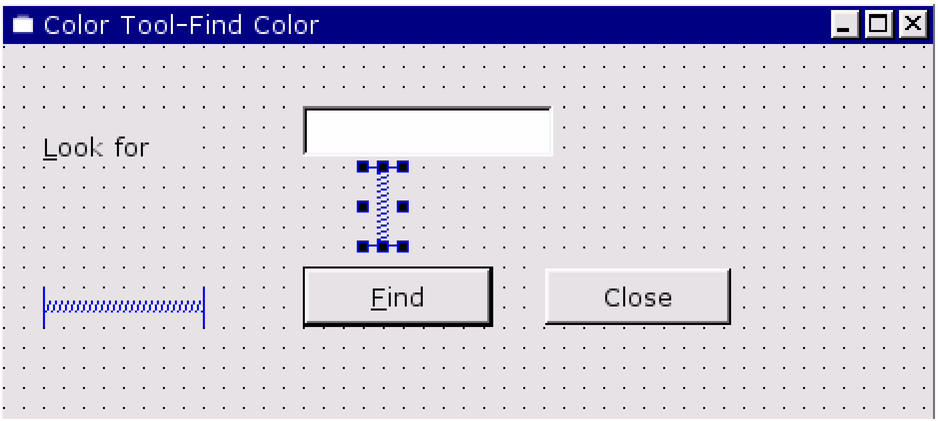


그림 1-54. 창문부품들의 추가

- 창문부품의 배치

창문부품들을 창조하고 대충 배치하였으므로 이제는 그것들을 배치하여야 한다.

폼을 선택하여 창문부품의 선택을 해제한다. Close단추의 오른쪽 아래를 선택하고 선택창을 2개의 단추와 수축자에 닿도록 Find단추의 왼쪽으로 끌고가서 놓는다. 너무 정확하지 않아도 된다. 오직 우리가 관심을 가지는 3개의 항목들에 이르기만하면 된다. 잘못하여 아무것도 선택되지 않거나 다른것을 선택하였다면 폼을 선택하고 다시 시도한다. 일단 수축자(Find단추의 왼쪽에)와 두 단추가 선택되면 Lay Out Horizontally도구띠단추를 찰칵한다.

Look for표식자과 행편집칸을 수평으로 배치한다. (Look for표식자를 선택하고 Shift를 누른 상태에서 행편집을 마우스로 선택하고 Ctrl+H를 누른다.)

이제는 폼자체를 배치할수 있다. 폼을 선택하고 Ctrl+L(수직배치)을 누른다. 폼의

크기를 약간 조절하여 더 작게 할수 있다. 폼을 미리보기(Ctrl+T)하고 크기를 조절한 다음 보기 좋은가 확인한다.

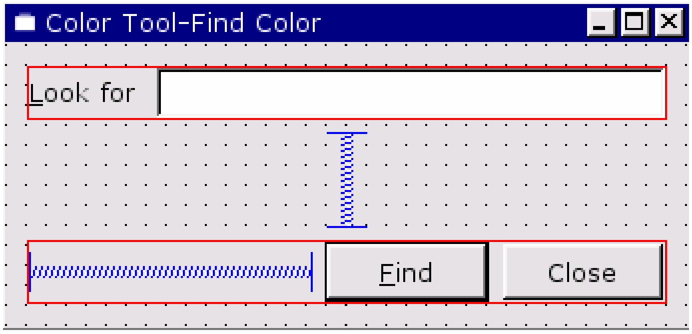


그림 1-55. 폼의 배치

- 창문부품의 련결

탐색대화칸을 만드는 2가지 수법이 있다. 하나는 이행금지대화칸을 사용하는 수법으로서 사용자가 단어를 입력하고 Find를 누르고 탐색한 항목을 강조표시하고 폼을 닫는다. 다른 하나는 이행허용대화칸을 사용하는 수법으로서 흔히 하는것처럼 사용자가 단어를 입력하고 Find을 누를 때마다 다음으로 일치하는 단어를 발견한다. 여기서는 둘째 수법을 사용한다.

기본폼에 보관된 자료를 통하여 탐색을 진행하고 발견한 단어를 기본폼에서 강조표시하려고 하므로 대부분의 탐색코드를 기본폼에 넣는다. 이것을 달성하려면 사용자가 Find단추를 누를 때마다 FindForm에 신호를 발송하여 일치하는 색을 발견하지 못했다는것을 FindForm에 통지하기 위하여 기본폼이 호출하는 처리부를 제공한다.

·View and Edit Connections대화칸을 펼친다. (Edit|Connections을 찰칵한다.)

·closePushButton의 clicked()신호를 폼의 accept()처리부와 련결한다. (New를 누르고 Sender를 closePushButton로, Signal을 clicked()로, 폼을 FindForm로, 처리부를 accept()로 변경한다.) 이 기능은 순수 신호-처리부련결을 통하여 얻어지므로 미리보기방식에서 작업한다. 즉 미리보기하고 Close단추를 찰칵하면 폼은 닫긴다.

·findPushButton의 clicked()신호를 새로 창조한 find()처리부로 변경한다. (New를 누르고 Sender를 findPushButton으로, Signal를 clicked()로, 폼을 FindForm으로 변경한다. Edit Slots를 눌러서 Edit Functions대화칸을 펼치고 처리부이름을 find()로 변경하고 OK를 찰칵한다. View and Edit Connections대화칸로 돌아와서 처리부를 새로 창조한 find()처리부로 변경한다.)

·View and Edit Connections대화칸을 닫는다. (OK를 찰칵한다.)

·사용자가 Find단추를 누를 때 기본폼이 본문을 찾을수 있도록 신호를 발생한다.

·Object Explorer의 Members타브를 선택하고 Signals을 오른쪽 단추로 찰칵하고 New를 누르고 lookfor(const QString&)라고 입력한다. find()처리부용 코드를 실현할

때 이 신호를 발생한다.

- 대화칸코드작성

Project Overview에서 findform.ui.h을 선택하여 코드편집기를 연다. find()함수를 실현하고 호출자가 탐색이 실패하였다는것을 알리는데 사용하는 notfound함수를 실현한다.

```
void FindForm::find()
{
    emit lookfor( findLineEdit->text() );
}
```

사용자가 Find단추를 누를 때 행편집에 있는 본문을 발생한다. 호출자에 따라서 이 신호와 련결하고 탐색을 처리한다.

```
void FindForm::notfound()
{
    findLineEdit->selectAll();
}
```

단어를 전혀 찾지 못하면 본문을 강조표시하고 그렇지 않으면 통보창문을 펼쳐야 한다.

대화칸의 코드는 기본폼에 모든 작업을 포함시키므로 간단하다.

2) 대화칸리용

사용자가 응용프로그람에서 Edit|Find를 선택하면 FindForm대화칸을 열고 Find단추를 누를 때마다 현재 보기에서 입력한 본문을 현재 작업중에 있던 색의 뒤에 있는 색으로부터 시작하여 탐색한다. FindForm의 바로 하나의 실례를 생성하고 그것에로의 지적자를 보유함으로써 필요할 때 그것을 표시하고 은폐할수 있다.

FindForm를 포함하고 init()함수에 (설명문으로) 넣은 findForm변수를 선언한다.

Project Overview에서 mainform.ui.h를 찰칵한다. 이때 코드편집기가 열리고 Object Explorer를 MainForm안의 객체들을 표시하도록 설정한다.

Includes(in Declaration)에 findform.h를 추가한다.(Object Explorer의 Members타브를 선택하고 Includes(in Declaration)를 오른쪽 단추로 찰칵하고 New를 누르고 findform.h라고 입력한 다음 Enter를 누른다.)

이제는 선언의 머리부에 다음의 선언을 추가한다.

"findform.h"

클라스변수에 FindForm *findForm;을 추가한다. (Class Variables를 오른쪽 단추로 선택하고 Edit를 누르고 Add를 찰칵한다. FindForm *findForm;라고 입력하고 OK를 찰칵한다.)

다음으로 클라스변수에 다음의 변수를 추가한다.

FindForm *findform;

init()함수에서 행 findForm = 0;의 설명을 해제한다.

이제는 editFind()처리부를 실현할수 있다.

- editFind()

```
void MainForm::editFind()
{
    if ( ! findForm ) {
        findForm = new FindForm( this );
        connect( findForm, SIGNAL( lookfor(const QString&) ),
                 this, SLOT( lookfor(const QString&) ) );
    }
    findForm->show();
}
```

FindForm을 생성하지 않았다면 그것을 생성하고 lookfor()신호를 기본폼에서 창조한 대응하는 lookfor()처리부와 련결한다. 그다음 FindForm을 표시하여 사용자가 탐색본문을 입력하고 탐색을 시작한다.

- lookfor()

```
void MainForm::lookfor( const QString& text )
{
    if ( text.isEmpty() )
        return;
    QString ltext = text.lower();
    QWidget *visible = colorWidgetStack->visibleWidget();
    bool found = FALSE;

    if ( visible == tablePage && colorTable->numRows() ) {
        int row = colorTable->currentRow();
        for ( int i = row + 1; i < colorTable->numRows(); ++i )
            if ( colorTable->text( i, 0 ).lower().contains( ltext ) ) {
                colorTable->setCurrentCell( i, 0 );
                colorTable->clearSelection();
                colorTable->selectRow( i );
                found = TRUE;
```

```
            break;
        }
        if ( ! found )
            colorTable->setCurrentCell( row, 0 );
    }
    else if ( visible == iconsPage ) {
        QIconViewItem *start = colorIconView->currentItem();
        for(QIconViewItem *item = start->nextItem(); item; item = item->nextItem())
            if ( item->text().lower().contains( ltext ) ) {
                colorIconView->setCurrentItem( item );
                colorIconView->ensureItemVisible( item );
                found = TRUE;
                break;
            }
        if ( ! found && start )
            colorIconView->setCurrentItem( start );
    }
    if ( ! found ) {
        statusBar()->message( QString( "Could not find '%1' after here").arg( text ) );
        findForm->notfound();
    }
}
```

이 처리부는 사용자가 FindForm에서 Find단추를 찰칵할 때 호출된다. FindForm의 행편집에 사용자가 입력한 본문은 text파라메터로서 넘긴다. 본문이 없으면 단순히 되돌아간다.

대소문자구별하여 탐색하려고 하므로 본문의 소문자사본을 가진다. 사용자가 어느 보기를 리용하고있는가 알아보고 found기발을 설정하고 후에 그 기발을 사용한다.

사용자가 표보기를 사용한다면 현재 작업하고있는 행의 다음행부터 탐색을 시작한다. 일치하는 본문을 찾으면 그것을 포함하는 행을 선택하고 found를 TRUE로 설정하고 탐색을 중지한다. 일치하는 본문을 찾지 못하면 현재 세포를 탐색을 시작한 세포뒤로 설정한다.

사용자가 그림기호보기를 사용한다면 현재 항목의 다음 항목부터 탐색을 시작한다. 일치하는 본문을 찾으면 대응하는 항목을 선택하고 그것을 볼수 있게 한다. 다시 일치하는 본문을 찾지 못하면 현재 항목을 탐색을 시작한 항목으로 설정한다.

본문을 발견하면 관련항목은 선택되였으므로 사용자의 보기에서 강조표시된다. 본문을 발견하지 못하면 상태띠에 통보를 표시하고 FindForm의 (탐색본문을 단지 선택하는) notfound()함수를 호출한다.

코드편집기에 직접 입력한 함수들은 (그것들이 공개처리부로 되는 경우에 돌림값이 void아니면) 공개함수로 된다. 이것들은 후에 함수속성을 편집하여 변경할수 있다. lookfor()는 그것과 련결해야 하므로 처리부로 되여야 한다. Object Explorer의 Members타브를 선택하고 lookfor()를 오른쪽 단추로 찰칵하고 Properties를 찰칵한다. 이것은 Edit Functions대화칸을 연다. Type를 slot로 변경하고 OK를 찰칵한다.

응용프로그람을 보존(Ctrl+S)하고 건설하고 탐색을 시도해본다.

3. 사용자선택

표보기에서 웨브색인가를 가리키는 선택과 Ctrl+C를 눌러 색을 복사할 때 오려둠판에 복사하는 선택을 사용자에게 주려고 한다. 또한 보기, 창문크기, 위치를 자동적으로 보존하고 되살리려고 한다.

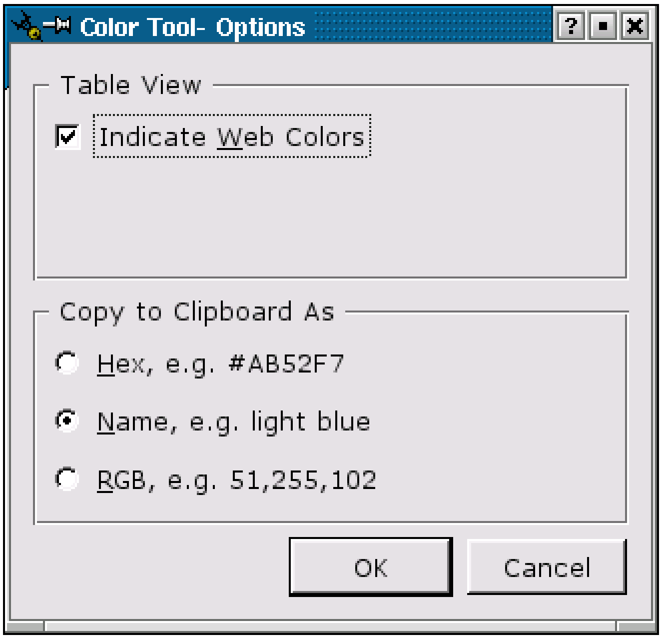


그림 1-56. 색속성설정도구

1) 대화칸작성

새 대화칸을 창조한다. (File|New를 눌러서 New File대화칸을 열고 Dialog를 선택하고 OK를 찰칵한다.) 폼의 name속성을 OptionsForm으로, caption속성을 "Color Tool -- Options"로 변경한다. 폼의 모서리를 끌어서 폼을 좀 작게 만든다. 이제 폼을 보관하고 기정값을 받아들인다.

Containers도구칸에서 GroupBox도구를 선택하고 폼의 왼쪽웃부분을 찰칵한다.

그룹칸을 끌기하여 그 폭을 늘인다. title속성을 Table View로 변경한다.

Common Widgets도구칸에서 CheckBox도구를 선택하고 그룹칸안을 찰칵한다. 검사칸의 name을 webCheckBox로, text속성을 Indicate &Web Colors로, checked속성을 True로 변경한다.

지금까지는 항상 모든 창문부품들을 배치하고 그것을 배치하였다. 그러나 여기서는 창문부품들을 배치하면서 배치한다. 그룹칸을 선택하고 Ctrl+H(수평배치)를 누른다.

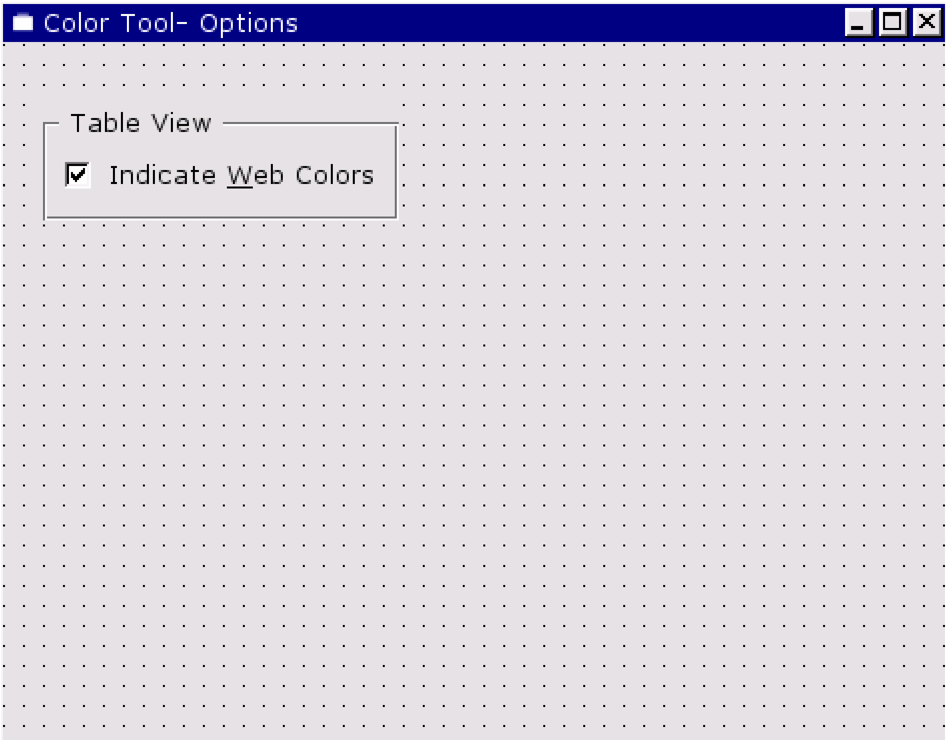


그림 1-57. 검사칸의 추가

ButtonGroup도구를 선택하고 표보기그룹칸아래를 찰칵한다. 그룹의 title속성을 Copy to Clipboard As로 변경하고 모서리를 끌기하여 더 크게 만든다.

오려둠판그룹에 3개의 라지오단추를 추가하려고 한다. 여러개의 같은 창문부품들을 추가할 때 Qt Designer의 다중배치방식을 사용하여 더 빨리 작업할수 있다.

RadioButton도구를 련속 두번 찰칵하여 다중배치방식에 들어간다. 그러면 폼을 누를 때마다 라지오단추가 하나씩 생성된다. 오려둠판그룹칸의 안과 우를 찰칵한다. 첫 라지오단추아래를 찰칵한다. 세번째로 둘째 라지오단추의 아래를 찰칵한다. Pointer도구띠단추를 찰칵하여 다중배치방식을 취소한다.

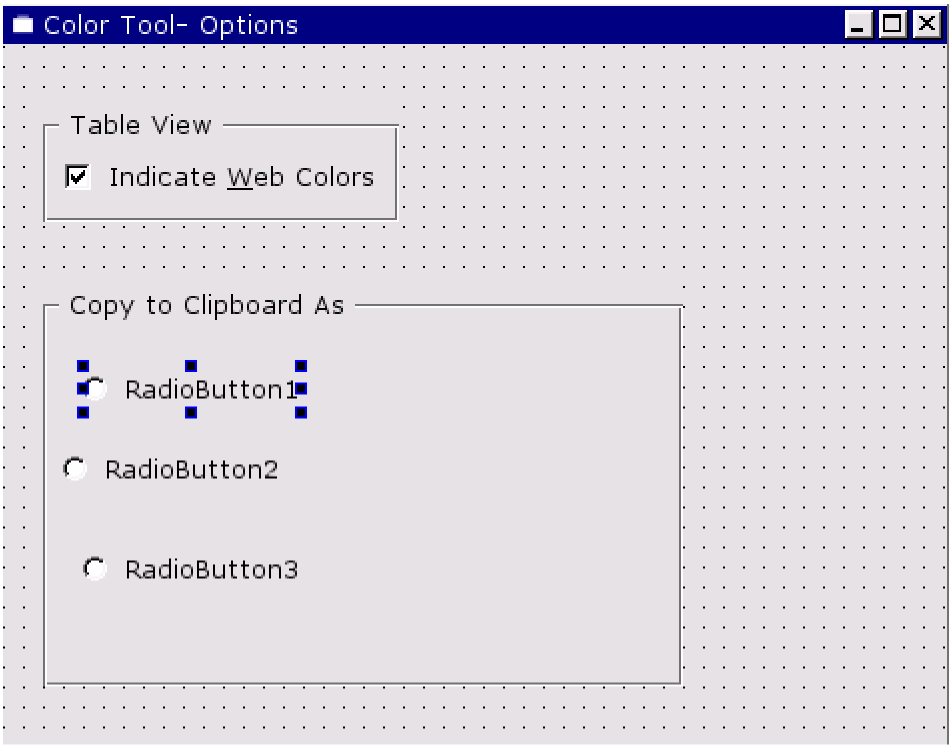


그림 1-58. 라지오단추의 추가

첫째(제일 우에 있는) 라지오단추를 찰칵하고 name속성을 hexRadioButton로, text속성을 &Hex, e.g. #AB52F7로, checked속성을 True로 변경한다.

둘째(중간) 라지오단추를 찰칵하고 name속성을 nameRadioButton로, 본문을 &Name, e.g. light blue로 변경한다.

셋째(아래) 라지오단추를 찰칵하고 name속성을 rgbRadioButton로, 본문을 &RGB, e.g. 51,255,102로 변경한다.

오려둡판그룹을 선택하고 Ctrl+L(수직배치)를 누른다.

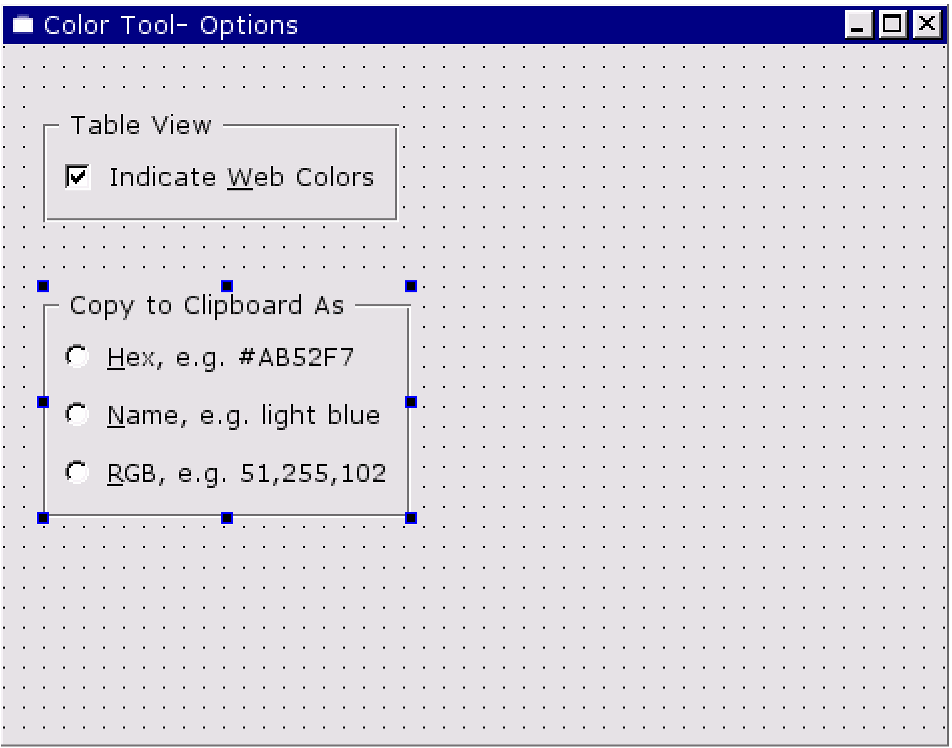


그림 1-59. 오려둠판그룹의 수직배치

이제는 OK단추와 Cancel단추를 생성한다. (PushButton도구를 선택하고 오려둠판그룹아래의 폼을 찰칵한다. 단추의 이름을 okPushButton로, text를 OK로, default속성을 True로 설정한다. PushButton도구를 다시 선택하고 OK단추의 오른쪽을 찰칵한다.단추의 이름을 cancelPushButton으로, 본문을 Cancel로 변경한다.)

단추들을 배치하기 위하여 OK단추의 왼쪽에 수평수축자를 추가하여 수축자와 단추들을 수평으로 배치한다. (Spacer도구를 선택하고 OK단추의 왼쪽의 폼을 누르고 좀 왼쪽으로 수평으로 끌고가서 놓는다. Shift를 누른 상태에서 OK단추를 누르고 Shift를 누른 상태에서 Cancel단추를 찰칵한다. Ctrl+H를 누른다.)

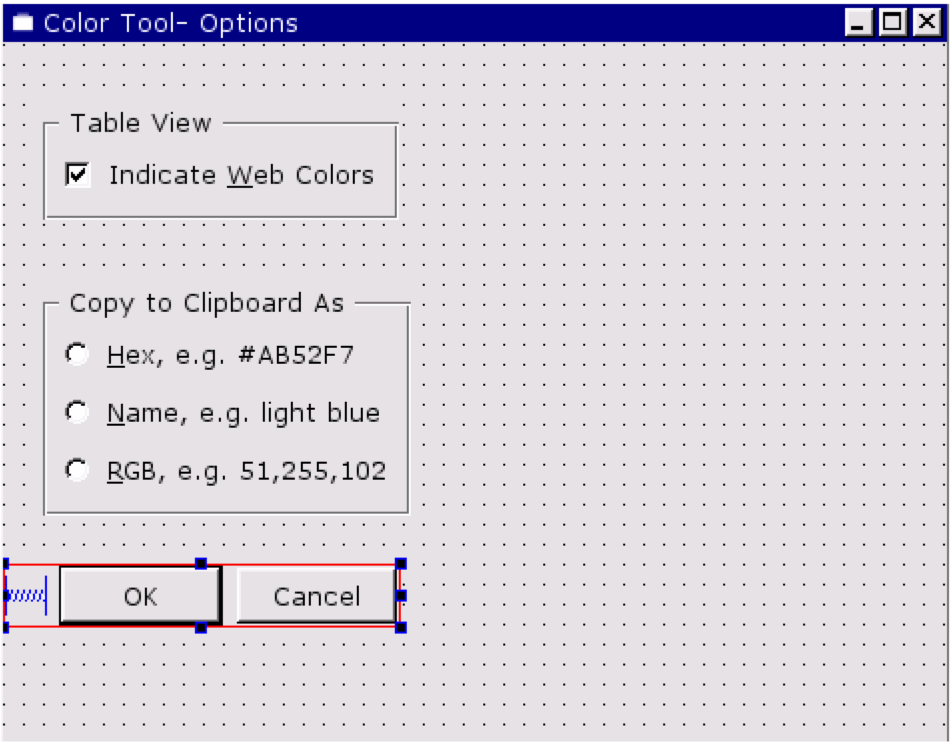


그림 1-60. 단추와 수평수축자의 추가

폼을 배치한다. (폼을 선택하고 Ctrl+L을 눌러 수직으로 배치한다.) 폼의 크기를 조절하여 적당한 크기와 모양으로 만든다.

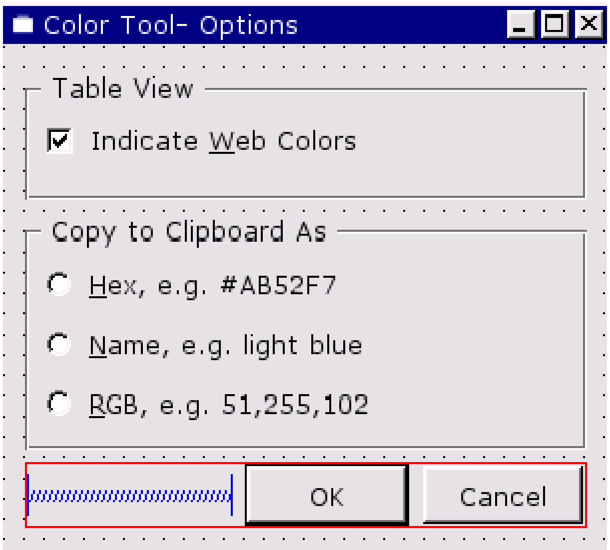


그림 1-61. 폼의 수직배치

- 창문부품의 련결

options대화칸은 전통적인 대화칸로서 호출자는 그것을 창조하고 창문부품들을 설정하고 사용자가 OK를 찰칵하면 폼의 창문부품들로부터 자료를 읽어들이고 동작한다. 해야 할 작업은 OK와 Cancel단추를 련결하는것이다. 코드를 쓸 필요는 없다.

OK단추를 폼의 accept()처리부에, Cancel단추를 폼의 reject()처리부에 련결한다. (Edit|Connections를 선택하고 New를 찰칵하고 Sender를 okPushButton으로, Signal을 clicked()로, Receiver를 OptionsForm으로, 처리부를 accept()로 변경한다. 다시 New를 찰칵하고 Sender를 cancelPushButton으로, Signal을 clicked()로, Receiver를 OptionsForm으로, 처리부를 reject()로 변경한다. OK를 찰칵하여 련결대화칸을 닫는다.)

2) 대화칸의 리용

Project Overview에서 mainform.ui.h를 눌러서 코드편집기를 호출하고 MainForm을 Qt Designer의 현재폼으로 설정한다.

options폼을 호출하는 처리부를 창조하고 이 처리부에 optionsAction작용을 련결해야 한다.

View and Edit Connections대화칸을 열고 optionsAction으로부터 editOptions라는 새 처리부에로의 새 련결을 창조한다. (Edit|Connections를 선택하고 New를 찰칵한다. Sender를 optionsAction로, Signal를 activated()로, Receiver를 MainForm으로 변경한다. Edit Slots단추를 찰칵하여 Edit Functions대화칸을 연다. New Function을 누르고 처리부의 이름을 editOptions()로 변경하고 OK를 찰칵한다. Slot함수를 새로 창조한 editOptions()처리부로 만든다. OK를 찰칵한다.)

이제는 처리부의 코드를 작성할 준비가 되였다.

- editOptions()

```
void MainForm::editOptions()
{
    OptionsForm *options = new OptionsForm( this, "options", TRUE );
    switch ( m_clip_as ) {
    case CLIP_AS_HEX:
        options->hexRadioButton->setChecked( TRUE );
        break;
    case CLIP_AS_NAME:
        options->nameRadioButton->setChecked( TRUE );
        break;
    case CLIP_AS_RGB:
```

```
        options->rgbRadioButton->setChecked( TRUE );
        break;
    }
    options->webCheckBox->setChecked( m_show_web );

    if ( options->exec() ) {
        if ( options->hexRadioButton->isChecked() )
            m_clip_as = CLIP_AS_HEX;
        else if ( options->nameRadioButton->isChecked() )
            m_clip_as = CLIP_AS_NAME;
        else if ( options->rgbRadioButton->isChecked() )
            m_clip_as = CLIP_AS_RGB;
        m_table_dirty = m_show_web !=
                    options->webCheckBox->isChecked();
        m_show_web = options->webCheckBox->isChecked();

        populate();
    }
}
```

새로운 선택폼을 창조하고 거기에 TRUE를 넘기여 모달로 만든다. m_clip_as변수의 현재설정에 따라 라지오단추를 설정한다. m_show_web변수와 대응되게 검사칸을 설정한다. 폼을 실행하고 사용자가 OK를 찰칵하면 그 선택을 련관된 기본폼변수들에 반영한다. 사용자가 webCheckBox를 눌러서 m_show_web변수를 변경하면 갱신을 요구하므로 표를 변경된것으로 표식한다. 그다음 필요하다면 표보기를 갱신하는 populate()를 호출한다.

OptionsForm을 사용하여 그 라지오단추와 검사칸을 호출하므로 실현의 머리부들에 optionsform.h와 qradiobutton.h, qcheckbox.h를 추가해야 한다. (Object Explorer의 Members타브를 선택하고 Includes(in Implementation)를 오른쪽 단추로 누른 다음 Edit를 찰칵한다. Add를 누르고 optionsform.h라고 입력하고 다시 Add를 누르고 qradiobutton.h라고 입력한다. 다시 Add를 누르고 qcheckbox.h라고 입력한다. Enter를 누른 다음 Close를 선택한다.)

실현의 머리부들에는 다음의 선언들이 추가되여있다.

"optionsform.h"

"qcheckbox.h"

"qradiobutton.h"

그러면 사용자는 자기 기호에 맞게 선택을 변경할수 있다. 그러나 이 선택설정은 응용프로그람을 완료할 때 잃어버린다. 사용자의 설정을 적재하고 보관하는 함수들을 추가하는것으로 끌마친다.

4. 환경설정의 보관과 적재

론리적으로 응용프로그람이 기동할 때의 적재설정과 응용프로그람완료시에 보관설정을 고찰한다. 그러나 우선 보관설정코드를 작성한다. 그래야 적재해야 할 내용을 알수 있다.

Qt 3.0은 새 클라스 QSettings를 받아들여 폼독자적인 방법으로 사용자의 설정을 조종한다. (례하면 Windows에 대한 등록과 Unix에 대한 rc파일들을 사용한다.) 실현파일들의 머리부에 qsettings.h머리부파일을 추가한다. (Object Explorer의 Members타브를 선택하고 "Includes (in Implementation)"를 오른쪽 단추로 찰각하고 New를 누르고 qsettings.h를 입력하고 Enter를 누른다.)

실현파일의 머리부에 다음의 선언을 추가한다.

"qsettings.h"

1) saveSettings()

```
void  MainForm::saveSettings()
{
    QSettings  settings;
    settings.insertSearchPath(  QSettings::Windows,  WINDOWS_REGISTRY  );
    settings.writeEntry(  APP_KEY  +  "WindowWidth",  width()  );
    settings.writeEntry(  APP_KEY  +  "WindowHeight",  height()  );
    settings.writeEntry(  APP_KEY  +  "WindowX",  x()  );
    settings.writeEntry(  APP_KEY  +  "WindowY",  y()  );
    settings.writeEntry(  APP_KEY  +  "ClipAs",  m_clip_as  );
    settings.writeEntry(  APP_KEY  +  "ShowWeb",  m_show_web  );
    settings.writeEntry(  APP_KEY  +  "View",
          colorWidgetStack->visibleWidget()  ==  tablePage  );
}
```

insertSearchPath()호출이 모든 가동환경에서 있어야 한다. (이 함수는 그것을 적용하지 못하는 가동환경에서 호출되면 단순히 되돌아간다.) 기본폼의 창문크기와 오려둠판용 선택, 웨브색지시기들을 보관한다. 또한 사용자의 보기를 기록한다. 사용자가 응용프로그람을 완료할 때 이 함수를 호출하므로 이제는 fileExit()함수의 제3판과 최종판을 만든다.

- fileExit()

```
void  MainForm::fileExit()
```

```
{
   if ( okToClear() ) {
      saveSettings();
      QApplication::exit( 0 );
   }
}
```

완료가 발생하면 자동적으로 사용자의 설정을 보관한다.

2) loadSettings()

```
void MainForm::loadSettings()
{
   QSettings settings;
   settings.insertSearchPath( QSettings::Windows, WINDOWS_REGISTRY );
   int windowWidth=settings.readNumEntry(APP_KEY +"WindowWidth",550);
   int windowHeight = settings.readNumEntry(APP_KEY+ "WindowHeight",
                                            500 );
   int windowX = settings.readNumEntry( APP_KEY + "WindowX", 0 );
   int windowY = settings.readNumEntry( APP_KEY + "WindowY", 0 );
   m_clip_as = settings.readNumEntry( APP_KEY + "ClipAs",CLIP_AS_HEX);
   m_show_web = settings.readBoolEntry( APP_KEY + "ShowWeb",TRUE );
   if ( ! settings.readBoolEntry( APP_KEY + "View", TRUE ) ) {
      colorWidgetStack->raiseWidget( iconsPage );
      viewIconsAction->setOn( TRUE );
   }
```

설정이 없으면 기정값을 리용하여 설정에 읽어들인다. (즉 설정이 삭제되였거나 이것이 처음이면 사용자는 응용프로그람을 실행한다.) 다시 insertSearchPath()를 모든 가동환경에서 호출한다.

설정에 따라서 마지막으로 프로그람을 완료할 때 사용자가 보고있던 보기로 절환한다. 또한 기본창문을 마지막으로 사용한 크기와 위치로 조절한다.

init()함수에서 loadSettings();행의 설명을 해제한다.

응용프로그람을 건설하고 실행한다. 그림기호보기를 변경하고 options대화칸의 1개이상의 선택을 변경한다. 창문을 이동하고 크기를 조절해본다. 응용프로그람을 완료하였다가 다시 기동한다. 보기와 창문보기, 위치, 기타 선택들이 모두 되살아난다.

5. 결론

이제는 colortool응용프로그람을 완성하였다. 프로그람의 기능을 더 확장할수 있다.

실례로 표보기에 보충적인 렬들을 추가하여 RGB값(0..255), 비례RGB값(0.00..1.00), HSV값 등을 표시할수 있다.

Find대화칸에 대소문자를 구별하는 검사칸을 제공하고 그것을 사용할수 있게 코드를 갱신할수 있다.

내부색선택대화칸와 같은 기능을 제공하지만 사용자가 색을 이름짓게 하는 자체의 색선택대화칸을 생성한다.

Qt Designer의 기본우점은 폼을 빠르면서도 간단히 설계(재설계)할수 하는것이다. 창문부품배치는 간단한 두단계의 과정이다. 즉 2개이상의 창문부품 혹은 배치들을 선택한 다음 그것들에 배치(수직, 수평 혹은 살창)를 적용한다. 배치가 제대로 되지 않으면 Ctrl+Z를 눌러서 취소하거나 배치를 선택하고 Ctrl+B(배치해체)를 누른다. Qt Designer는 무제한한 취소(undo)와 재시행(redo)를 지원하므로 실험하기 쉽고 안전하다. (Qt Designer의 배치를 코드로 콤파일하는 방법을 알려면 응용프로그람을 건설할때 생성되는 .cpp파일들을 보시오.)

Qt Designer의 배치들을 시험해볼것을 권고한다. 실례로 multiclip실례(qt/tools/designer/examples/multiclip에서)의 보기와 크기조절특성들을 가지는 폼을 되살려보시오.

6. 오유

오유원천은 머리부파일, 앞방향선언 혹은 변수를 틀리게 입력하는것이다. 다음의 목록에 대하여 그것들을 검사하고 적당히 삽입하거나 수정한다. (모든 경우에 Object Explorer의 Members타브를 검사한다. 편집해야 한다면 련관된 부분을 오른쪽 단추로 선택하고 Edit를 눌러서 련관된 대화칸을 호출한다.)

1) MainForm성원들

- 클라스변수:

```
FindForm  *findForm;
QClipboard  *clipboard;
QMap<QString,QColor>  m_colors;
bool  m_show_web;
int  m_clip_as;
bool  m_icons_dirty;
bool  m_table_dirty;
bool  m_changed;
QString  m_filename;
QStringList  m_comments;
```

- 앞방향선언:

class QColor;

class QString;

- 선언에서 머리부파일들

"findform.h"

- 실현에서 머리부파일들:

"optionsform.h"

"qlineedit.h"

"qlabel.h"

"qclipboard.h"

"qmessagebox.h"

"qstatusbar.h"

"qpainter.h"

"qstring.h"

"qcolor.h"

"qapplication.h"

"qfiledialog.h"

"qfile.h"

"qregexp.h"

"qcolordialog.h"

"colornameform.h"

"qcheckbox.h"

"qradiobutton.h"

"qsettings.h"

2) ColorNameForm성원들

원천코드파일에 ColorNameForm선언들을 모두 넣는다. 파일 colornameform.ui.h는 다음의 선언들로 시작해야 한다.

```
#include <qcolor.h>
#include <qmap.h>
#include <qstring.h>
QMap<QString,QColor> m_colors;
```

3) FindForm성원들

신호:

lookfor(const QString&)

4) OptionsForm성원들

OptionsForm에는 성원이 없다.

5) main.cpp성원들

이 파일은 다음의 선언으로 시작해야 한다.

```
#include <qapplication.h>
#include "mainform.h"
```

제4절. Designer수법

Qt 2.x에서 Qt Designer는 .ui파일형식으로 파일들을 편집하기 위한 시각적인 폼설계프로그람이였다. Qt Designer의 본래 목적은 지루한 GUI프로그람작성부분(대화칸설계)을 재미있는 기능으로 전환하는것이였다. 2.x의 Qt Designer는 아주 간단한 프로그람으로서 .ui파일들을 읽고 쓴다. 각 .ui파일에는 단일한 대화칸폼의 XML서술이 들어있다. 둘째 봉사프로그람 즉 사용자대면부번역프로그람 uic는 XML서술로부터 C++코드를 생성하는 응용프로그람의 건설과정에 사용된다.

Qt 3.0에서 Qt Designer의 기능은 하나의 대화칸편집이상으로 높아졌다. 기본창문과 작용창조능력과 같은 새로운 설계특성들과 함께 새로운 판에서는 다음과 같은 기능을 받아들였다.

- 응용프로그람의 사용자대면부를 위한 프로젝트관리,
- 처리부의 코드를 직접 작성할수 있도록 Qt Designer가 코드편집기를 제공하는 폼들에서 코드작성. 코드는 .ui.h파일들에 보관되고 파생클라스를 작성할 필요가 없어졌다. (필요하다면 작성자가 파생클라스를 만들수 있다.)
- 실행시에 .u파일들을 적재하게 하는 동적폼적재. 이것은 기초코드와 분리된 설계전용화의 중요한 기회를 제공한다.

이 절의 목적은 이러한 변경방법을 설명하고 포함된 새로운 개념들을 서술하며 이 기능들의 내부적인 작업방법을 보여주는것이다.

Qt Designer는 시각적인 설계도구이지 완전한 통합개발환경이 아니다. 그 목적은 어떤 특별한 도구에 사용자들이 포로되지 않고 될수록 간단하고 강력한 GUI개발을 가능하게 하는것이다. Qt Designer는 GUI설계를 창조하고 수정하기 쉽게 하지만 아직도 필요하다면 일반본문편집기를 사용하여 같은 결과를 직접 코드로 얻을수 있다.

더 편리하게 하기 위하여 현재 Qt Designer에는 C++편집기를 플라그인으로서 포함한다. 폼을 생성하거나 편집하려면 Qt Designer를 사용해야 한다. 폼의 코드를 편집하려면 물론 Qt Designer의 C++편집기를 사용할수 있다. 이 기본편집기는 후에 설명하는 시각적인 폼설계과정과의 엄격한 통합으로부터 분리할수 있게 한다. 그러나 vim, emacs, notepad, Microsoft Visual Studio 등의 편집기를 사용하는것이 좋으면 여전히 그것을

사용할수 있다.

1. 프로젝트관리

하나의 비련결된 .ui파일을 읽고쓰는 작업은 개념적으로 단순하고 Qt 2.x에서 잘된다. 그러나 Qt 2.x에는 Qt Designer로 응용프로그람의 GUI부분에 대한 프로젝트관리를 받아들이게 한 일정한 특성들이 부족하였다. 프로젝트관리의 기본우점은 다음과 같다.

- 공통소유하는 폼들의 그룹화.
- 각이한 폼들사이에서 화상의 공유.
- 각이한 폼들사이에서 자료기지정보의 공유.

다음에 그 우점과 프로젝트관리의 필요성에 대하여 더 자세히 설명한다.

1) 폼의 그룹화

폼의 그룹화는 Qt Designer가 같은 프로젝트에 포함되는 .ui파일들의 목록을 관리한다는것을 의미한다. 이것은 한번의 마우스누르기로 폼들사이를 쉽게 절환하게 한다.

2) 화상집합에서 화상공유

Qt 2.x의 Qt Designer에서 매개 폼이 자기의 필요한 화상들을 포함하였으며 화상들의 공유는 없었다. 이것은 여러개의 폼이 같은 화상들을 사용해야 할 때 중복이 생기게 하였다. 게다가 화상들은 XML .ui파일들에 보관하였으므로 파일의 크기가 늘어났다.

그리하여 Qt Designer에서 정의할수 있는 픽스매프적재기능을 받아들였다. 이 수법의 큰 결함은 Qt Designer에서 설계과정에 화상들을 볼수 없는것이였다. 이것은 폼을 시각적으로 설계할수 없을뿐아니라 기하학적관리에도 현저한 영향을 주었다.

Qt 3.0판의 Qt Designer에서는 프로젝트화상집합이라는 개념을 도입하였다. 프로젝트를 사용한다면 프로젝트의 화상집합에 화상들을 추가할수 있으며 이 화상들을 공유하고 프로젝트에 포함한 임의의 폼들에서 사용할수 있다. 화상은 프로젝트의 등록부에 있는 보조등록부 images/에 PNG(portable network graphics)로 보관된다. 화상집합을 수정할 때마다 Qt Designer는 2진형식의 화상자료와 화상실례를 작성하는 함수를 둘다 포함하는 원천파일을 생성한다. 화상은 프로젝트의 모든 폼들에서 호출할수 있으며 자료는 공유된다.

화상집합사용의 다른 우점은 화상들이 기정의 QMimeSourceFactory에 추가되는것이다. 이리하여 리치본문표식자으로부터 호출할수 있다. 상황의존방조와 지어는 도구암시도 표준HTML화상꼬리표를 포함한다. 화상꼬리표의 source인수는 단순히 화상집합의 화상의 이름이다. 이것은 또한 Qt Designer에 의한 설계과정에 작업한다.

3) 공유자료기지설정

Qt 3.0은 새로운 자료기지모듈 Qt SQL모듈을 받아들였다. Qt Designer는 SQL모듈과 완전히 통합되며 련결된 자료기지로부터 생자료를 보여줄수 있다.

프로젝트를 열거나 창조했을 때 Project|Database Connections차림표선택에 의하여 펼쳐지는 Edit Database Connections대화칸을 사용하여 자료기지련결을 설정할수

있다. 작성자가 만드는 련결은 .db파일에 보관된다. 프로젝트를 재적재할 때 Edit Database Connections대화칸로 가서 련결목록을 선택하고 Connect단추를 찰칵하여 재련결할수 있다.

대부분의 자료기지응용프로그람에서는 하나이상의 폼으로부터 자료기지를 호출하려고 한다. 이것은 .db파일이 단일폼의 부분이 아니라 프로젝트의 부분이기때문이다.

4) .pro파일

Qt Designer는 프로젝트에 대한 정보 실례로 폼목록과 화상집합, 사용하는 자료기지와 그 호출방법에 대한 정보를 보관하여야 한다. 대부분의 Qt사용자들은 이미 프로젝트파일형식을 리용하여 다중가동환경 makefile을 만든다. 즉 tmake(와 Qt 3.0 qmake)프로젝트 .pro파일들이다. 이 파일들은 이미 폼들의 목록을 포함하는 .ui파일들로서 프로젝트에서 uic용으로 사용된다.

.pro파일안의 절들을 Qt Designer가 프로젝트를 관리하는데 필요한 여유정보를 포함하도록 확장하였다. 실례로 Qt Designer에서 프로젝트에 폼을 추가할 때 프로젝트파일의 FORMS절에 자동적으로 추가되므로 qmake는 다른 작업을 하지 않고도 필요한 건설규칙을 생성한다. 마찬가지로 화상들은 IMAGES절에 추가되므로 실행파일로 자동적으로 콤파일된다.

qmake사용을 강요하지 않는다. 다른 건설체계 실례로 automake/autoconf 또는 jam을 좋아한다면 그것을 계속 사용할수 있다. 응용프로그람의 GUI부분을 서술하는 파일로서 .pro파일을 간주한다. 필요한것은 자기의 Makefile들에 .ui파일들과 화상집합을 추가하는것이다.

2. 폼의 기능확장

우선 .ui파일들과 생성된 코드, 응용프로그람코드사이의 관계를 보여주는 그림 1-62를 고찰하자.

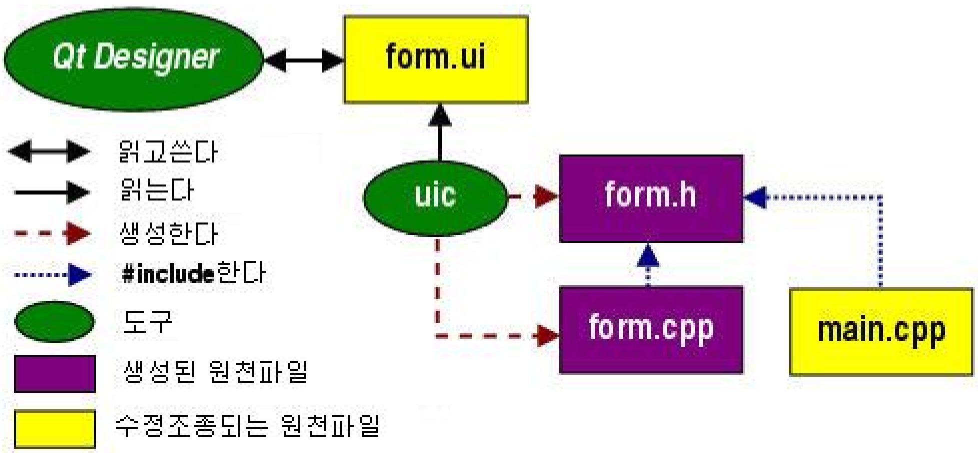


그림 1-62. .ui파일과 생성된 코드, 응용프로그람코드사이의 관계

Qt Designer는 .ui파일 실례로 form.ui를 읽고쓴다. 사용자대면부번역프로그람 uic는 머리부파일 실례로 form.h와 실현파일 실례로 form.cpp를 .ui파일로부터 둘다 창조한다. main.cpp의 응용프로그람코드는 form.h를 포함한다. 일반적으로 main.cpp는 QApplication객체의 실례를 만들고 사건순환고리를 시작하는데 사용된다.

이 수법은 간단하지만 더 복잡한 대화칸에는 충분하지 않다. 복잡한 대화칸은 폼의 창문부품들에 련결된 아주 많은 론리를 가지며 보통 미리 정의된 신호와 처리부들로 보여주는것보다 더 많은 론리를 가지고있다. 이 여유론리를 조종하는 하나의 방법은 폼에 기능을 추가하는 조종자클라스를 응용프로그람코드에 쓰는것이다. 이것은 uic가 생성한 클라스들이 공개부분에 폼의 조종과 그 신호를 내주기때문에 가능하다. 이 수법의 큰 결함은 그것이 정확한 Qt형식이 아닌것이다. Qt Designer를 사용하지 않는다면 대체로 폼자체에 론리를 추가하군 한다.

이것은 폼에 전용처리부들과 성원변수들을 추가하는 능력이 초기에 Qt Designer에 추가되였기때문이다. 이 수법의 중요한 우점은 미리 정의된 처리부에 신호를 련결하는데 사용하던 도형방식으로 Qt Designer를 사용하여 신호를 전용처리부에 련결할수 있는것이다. 그다음 uic는 매개 전용처리부에 대하여 빈 그루터기(stub)를 생성된 form.cpp실현파일에 추가한다.

여기서 큰 문제는 그 전용처리부들에 전용실현코드를 어떻게 추가하는가 하는것이다. 생성된 form.cpp에 코드의 추가는 선택적인것이 아니며 이 파일은 폼이 변경될 때마다 uic에 의하여 다시 생성되며 작성자들은 생성된 코드와 손으로 쓴 코드의 결합을 바라지 않는다. 2가지 가능한 해결책이 있는데 다음에 학습한다.

1) 파생클라스작성방법

생성된 폼에 전용처리부를 실현하는 아주 명백한 방법은 그림 1-63에 보여준것처럼 C++계승을 리용하는것이다.

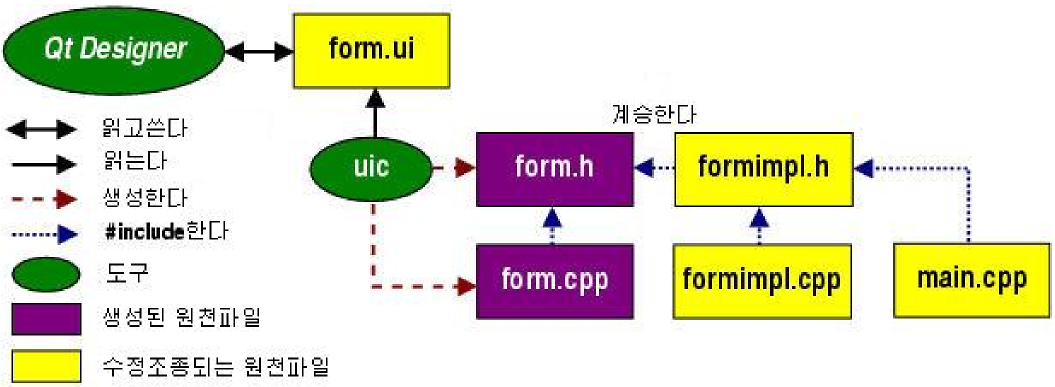


그림 1-63. 파생클라스작성수법

여기서 사용자는 보충적인 클라스 FormImpl을 작성하는데 머리부파일 formimpl.h

과 실현파일 formimpl.cpp로 가른다. 머리부파일은 uic가 생성한 form.h를 포함하며 모든 전용처리부들을 재실현한다. 이것은 uic가 생성한 전용처리부들이 가상이므로 가능하다. 전용처리부실현과 함께 이 수법은 파생클라스의 구성자에서 여분의 초기화를 수행하는 방법과 해체자에서 여분의 해체처리를 하는 방법을 사용자에게 준다.

이러한 우점과 그 유연성으로 하여 이 수법은 Qt 2.x에서 Qt Designer를 사용하는 원시적인 방법으로 되였다.

알아두기: 이름공간을 깨끗이 유지하기 위하여 대부분의 사용자들은 그림 1-63에서 보여준 Form과 FormImpl명명구조를 따르지 않았지만 그대신 Qt Designer의 폼들을 FormBase로, 파생클라스들을 Form으로 명명하였다. 이것은 폼들이 항상 파생클라스로 작성되고 응용프로그람코드에서 파생클라스들을 사용하고있으므로 리치적으로 옳았다.

2) ui.h확장수법

유연성과 완전성에도 불구하고 파생클라스작성수법은 일부 결함을 가진다.

• 파생클라스작성은 모두에게 자연스럽지 않고 쉽지 않다. 객체지향수법을 새로 배우는 사람들은 파생클라스로 작성하는것을 전용처리부의 실현보다 어려운 일로 생각한다.

• 생성된 클라스들을 계승하는것은 프로그람작성오유의 추가적인 원천이다. 특히 재실현함수의 수가 많고 설계과정에 기호가 자주 변경된다면 이것은 더하다. 개발과정을 더 원만하게 하기 위하여 uic는 순수가상함수보다도 전용처리부용 빈 그루터기를 생성한다. 이 수법은 코드의 콤파일 및 실행을 유지하는 한편 프로그람작성자들은 실행시경고오유통보문을 놓치는 상황에 부닥치게 되고 파생클라스에서 자그마한 철자오유를 발견하는데 시간을 소비한다.

• 수백개의 폼을 가지는 큰 프로젝트들에서 보충적인 파생클라스들은 콤파일속도와 코드크기에서 현저한 차이를 가진다.

몇가지 결함이 있지만 이것들은 다른 해결방법으로 충분히 조사할수 있는 원인이다. Qt 3.0에서는 새로운 개념 ui.h확장을 제공한다.

이것은 그림 1-64와 같이 작업한다.

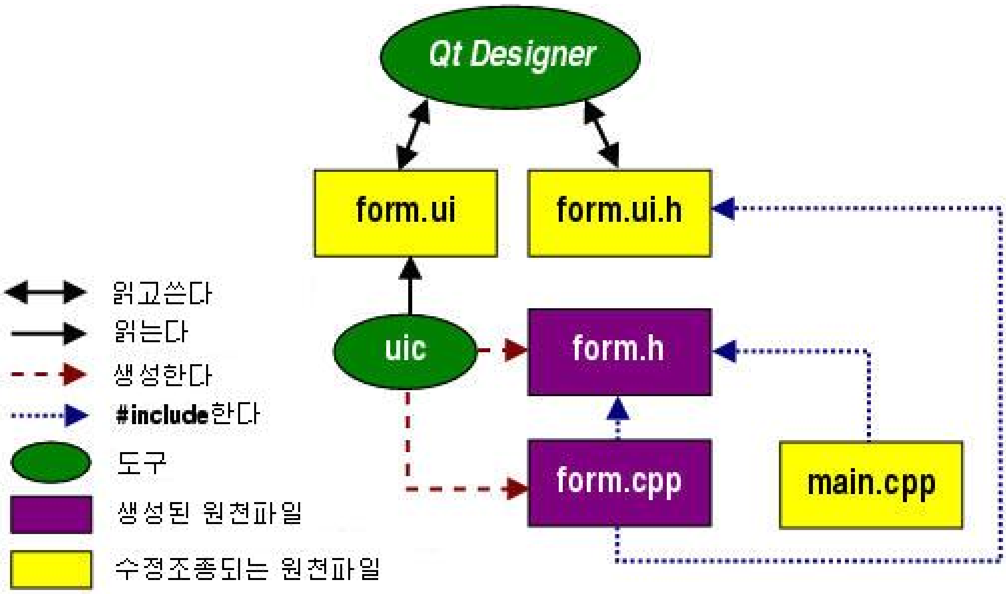


그림 1-64. .ui확장수법

.ui파일, form.ui와 함께 Qt Designer는 다른 련관된 파일 form.ui.h을 읽고 쓴다. 이 .ui.h파일은 보통의 C++원천파일로서 전용처리부의 실현을 포함하고 생성된 폼실현파일 form.cpp로부터 포함되므로 다른 사용자코드에서 완전히 무시될수 있다. C++코드를 포함하는 .h확장을 .ui.h파일에 사용하는 리유는 그것이 항상 포함되며 .h확장으로 건설과정에 통합하기 더 쉽기때문이다.

form.ui.h파일은 다른 모든 파일들중에서 특별한 위치를 차지한다. 이것은 사용자와 Qt Designer에 의하여 읽고씌워지는 공유원천파일로서 일반적으로 수정조종되는 원천파일이며 uic에 의하여 생성되지 않는다. Qt Designer가 할 일은 련관된 폼의 전용처리부 정의와 동기하여 파일들을 유지하는것이다.

① 사용자가 폼에 새 처리부를 추가할 때마다 Qt Designer는 .ui.h파일에 그루터기를 추가한다.

② 사용자가 전용처리부의 기호를 변경할 때마다 Qt Designer는 대응하는 실현을 갱신한다.

③ 사용자가 전용처리부를 삭제할 때마다 Qt Designer는 그것을 .ui.h파일에서 삭제한다.

이리하여 완전성이 담보되고 파생클라스를 만들 필요가 없어지고 파생클라스안의 처리부들을 잊어버리거나 철자가 틀릴 위험이 더는 없어진다.

.ui.h파일들은 내부C++편집기플라그인을 가지고있는 Qt Designer에서 직접 편집하거나 자기가 좋아하는 편집기에서 편집할수 있다. 오직 .ui.h파일에 처리부실현을 삽입하고 Qt Designer안에서 항상 처리부를 추가, 삭제하거나 이름변경하여야 한다. Qt

Designer에서 혹은 자기의 편집기를 사용하여 처리부의 실현을 편집할수 있다. 자체의 편집기를 사용한다면 Qt Designer는 변경을 보관한다.

- 구성과 해체

ui.h확장수법은 파생클라스작성수법에 비하여 한가지 결함을 가지고있다. ui.h파일은 오직 전용처리부실현만 포함하지만 객체는 여전히 생성된 form.cpp코드안에서 구성되고 해체된다. 이것은 보통 C++클라스의 구성자와 해체자에서 수행하는 폼의 초기화나 삭제를 사용자가 할수 없게 한다.

이러한 제한하에서 작업하기 위하여 init/destroy관례를 만들었다. 폼에 처리부 Form::init()를 추가하자면 이 처리부는 생성된 폼구성자의 끝에서 자동적으로 호출된다. 마찬가지로 폼에 처리부 Form::destroy()를 추가하면 처리부는 임의의 폼조종이 삭제되기전에 자동적으로 호출된다. (이 처리부들은 void를 돌려주어야 한다.) 자체의 편집기를 사용하기 좋아한다면 여전히 Qt Designer에서 이 함수들을 창조한다. Qt Designer의 C++편집기플라그인이나 자체의 편집기를 사용하여 실현코드를 쓸수 있다.

3. 폼의 동적적재

새로운 클라스 QWidgetFactory는 실행시에 .ui파일들을 적재하고 그것들로부터 폼의 실례를 만들수 있게 한다.

이 동적수법은 GUI설계와 독립적으로 코드를 유지하며 기초응용프로그람론리보다도 더 자주 GUI를 변경해야 하는 환경에서 쓸모있다. 끝으로 완전한 C++개발환경이 없이 도형방식사용자대면부를 수정할수 있는 능력을 응용프로그람의 사용자에게 제공할수 있다.

.ui파일은 콤파일되지 않으므로 C++코드(례를 들면 전용처리부실현)를 포함할수 없다. 창문부품factory에 수신자로서 넘기는 조종용QObject파생클라스를 통하여 그 실현을 추가하는 방법을 제공한다.

이 개념과 그 사용법은 다음 절에서 자세히 설명한다.

제5절. 파생클라스작성과 동적대화칸

이 절에서는 Qt Designer에 의하여 폼을 창조하는 두가지 수법을 설명한다. 파생클라스작성법(Subclassing)은 Qt Designer에서 창조하는 폼에 기초한 클라스를 자체로 창조함으로써 폼의 기능을 확장하는데 쓰인다. 동적대화칸은 Qt응용프로그람에 의하여 실행되는 .ui파일로서 GUI설계와 코드를 따로따로 보관하며 GUI를 기본응용프로그람론리보다 더 자주 변경시키는 환경에서 쓸모있다.

1. 파생클라스작성

폼의 파생클라스를 만드는 방법에 대하여 일반적으로 설명하고 간단한 실례로부터 시작한다. 파생클라스작성법은 폼에 직접 코드를 삽입하는 경우보다 몇가지 결함이 있다. (4절 2. 참고)

1) Qt Designer .ui파일로부터 원천코드의 생성

Qt Designer는 qmake의 .pro(프로젝트)파일을 읽어들이고 써넣는다. 이 파일은 응용프로그람을 건설하는데 쓰이는 파일들을 기록하는데 사용되고 그로부터 Makefile들을 생성한다. 또한 Qt Designer는 .ui(사용자대면부)파일들을 읽어들이고 써넣는다. .ui파일은 폼에 사용한 창문부품, 배치, 원천코드와 환경설정을 기록하는 XML파일이다. 매개의 .ui파일은 uic(사용자대면부콤파일러)에 의하여 C++의 .h파일과 .cpp파일로 변환된다. 그다음 이 C++파일들은 moc(메타객체콤파일러)에 의하여 읽어들이고 최종적으로 콤파일러에 의하여 작업응용프로그람으로 콤파일된다.

Qt Designer에서 응용프로그람을 완전히 작성한다면 하나의 main.cpp를 생성해야 한다.

Qt Designer안에서 main.cpp파일을 창조한다면 Qt Designer에 의하여 프로젝트파일에 자동적으로 추가된다. Qt Designer밖에서 main.cpp파일을 창조한다면 프로젝트의 .pro파일의 끝에 다음의 행을 추가하여 프로젝트파일에 수동적으로 추가해야 한다.

```
SOURCES += main.cpp
```

그다음 qmake를 사용하여 Makefile을 생성한다. (실례로 qmake -o Makefile myproject.pro.) make(Linux, Unix 또는 Borland번역프로그람)를 실행하거나 nmake(Visual C++)를 실행하면 uic, moc 그리고 콤파일러들이 필요할 때 호출되여 응용프로그람을 건설한다.

Qt Designer를 사용하여 기본창문과 대화칸을 생성하지만 다른 C++파일들을 추가하거나 또는 임의의 폼을 파생클라스작성한다면 .pro파일에 이 파일들을 추가하여 응용프로그람의 나머지 원천파일들과 함께 콤파일해야 한다. main.cpp에 대하여 수행한것처럼 Qt Designer로부터 개별적으로 창조하는 매개의 .h파일은 HEADERS행에 추가되고 매개 .cpp파일은 SOURCES행에 추가되여야 한다. 정의되지 않은 참고오유가 발생하면 모든 머리부와 실현파일들의 이름을 .pro파일에 추가하였는가 검사하는것이 좋다.

2) 폼의 파생클라스작성

폼의 파생클라스를 작성할 때 명명관례를 리용하여 Qt Designer의 .ui파일들로부터 생성되는 파일들과 손으로 코드작성한 파일들을 식별할수 있다.

실례로 Qt Designer에서 대화칸을 개발하고 직접 코드를 쓴다고 가정하자. 대화칸을 OptionsForm, .ui파일을 optionsform.ui라고 부르기로 하자. 자동생성된 파일들은 optionsform.h과 optionsform.cpp이다.

이번에는 다른 대화칸을 파생클라스로 만들려고 하는데 자동생성된 파일들과 손으

로 작성한 파일들을 쉽게 구별하려고 한다. 실례로 대화칸을 SettingsFormBase, .ui파일을 settingsformbase.ui라고 부르기로 한다. 그다음 자동생성된 파일들은 settingsformbase.h와 settingsformbase.cpp로 불리운다. 파생클라스를 SettingsForm라고 하고 파일 settingsform.h와 settingsform.cpp에 코드를 쓰려고 한다.

폼의 파생클라스는 Q_OBJECT마크로를 포함하여야 처리부와 신호들이 정확히 작업한다. 파생클라스를 생성한 다음에는 .h와 .cpp파일들을 .pro프로젝트파일에 추가하려는가 확인해야 한다. 실례로 파생클라스 SettingsForm를 위하여 .pro파일의 끝에 다음 행을 추가한다.

```
HEADERS += settingsform.h
SOURCES += settingsform.cpp
```

새로운 원천파일을 창조하는 가장 간단한 방법은 File|New를 선택하여 New File 대화칸을 열고 적당히 C++ Source 또는 C++ Header를 선택하고 OK를 찰칵한다. 빈 원천창문이 새로 표시된다. Qt Designer가 자동적으로 프로젝트파일에 추가하므로 .pro 파일을 수동적으로 편집할 필요가 없다.

Qt Designer는 프로젝트파일에 다음행을 추가한다.

```
FORMS = settingsformbase.ui
```

settingsformbase.h와 settingsformbase.cpp파일들은 .ui파일로부터 자동적으로 생성된다.

3) 파생클라스작성실례

작은 실례대화칸을 만들어 파생클라스작성을 실제로 보여준다. 대화칸은 손님의 대부률과 특정한 량으로 주어지는 특수비률의 선택을 표시한다. 이 기능을 파생클라스로 실현한다. 기본폼을 창조하고 그 신호와 처리부들을 련결하는것으로 시작하여 파생클라스와 간단한 main.cpp를 창조하여 시험할수 있다.

- 폼의 설계

새 프로젝트를 창조하는것부터 시작한다. File|New를 선택하고 C++ Project그림기호를 선택하여 Project Settings대화칸을 펼친다. 생략단추를 찰칵하여 Save As대화칸을 열고 프로젝트의 등록부(필요하다면 생성하여)로 이행한다. 프로젝트의 등록부를 확인한 다음 프로젝트의 이름을 credit.pro라고 입력한다. Save단추를 찰칵하여 Project Settings대화칸로 돌아와서 OK를 찰칵한다. 이제는 프로젝트에 폼을 추가한다. File|New를 선택하여 New File대화칸을 연다. OK를 찰칵한다. 폼의 크기를 작게 만들어 약 2in(5cm)의 정방형으로 만든다. 폼의 이름을 CreditFormBase로, 제목을 Credit Rating으로 변경한다. 폼을 creditformbase.ui로 보관한다.

이제는 필요한 창문부품들을 추가한다.

① Button Group도구띠단추를 찰칵하고 폼의 왼쪽웃구석근방을 선택한다. 단추그룹의 크기를 조절하여 폼의 약 절반으로 만든다. 단추그룹의 name을

creditButtonGroup로, title속성을 Credit Rating으로 변경한다.

② 라지오단추들을 추가한다. Radio Button도구띠단추를 두번 련속 찰칵한다. Credit Rating단추그룹의 꽉대기쪽을 선택하면 라지오단추가 하나 나타난다. 이 단추아래를 선택하여 둘째 라지오단추를 생성하고 둘째단추아래를 선택하여 셋째 단추를 생성한다. Pointer도구띠단추를 찰칵하여 두번 찰칵의 효과를 차단한다. 그러면 지시기가 정상으로 동작한다. 즉 폼을 선택하면 더는 라지오단추들이 생성되지 않는다. 첫째 라지오단추의 name을 stdRadioButton로, text를 &Standard로, checked속성을 True로 변경한다. 둘째 단추의 이름을 noneRadioButton으로, text속성을 &None으로 변경한다. 셋째 단추의 속성을 specialRadioButton으로, text속성을 Sp&ecial로 각각 변경한다.

③ 사용자가 특수대부률을 선택하면 량을 지정해야 한다. SpinBox도구띠단추를 찰칵하고 바로 단추그룹아래의 폼을 선택한다. 스핀단추의 name을 amountSpinBox로, prefix를 $(공백을 표시), maxValue를 100000로, lineStep을 10000으로, enabled속성을 False로 변경한다.

④ Push Button도구띠단추를 찰칵하고 스핀단추아래의 폼을 선택한다. 단추의 name을 okPushButton으로, text를 OK로, default속성을 True로 각각 변경한다. 첫째단추의 오른쪽에 둘째 단추를 추가하고 둘째 단추의 name을 cancelPushButton으로, text를 Cancel로 변경한다.

이제는 창문부품들을 배치하고 필요한 처리부들을 련결한다.

① 대부률그룹칸을 선택하고 Ctrl+L(수직배치)을 누른다.

② 폼을 선택하여 단추그룹이 더는 선택되지 않게 한다. Ctrl을 누르면서 OK단추를 누르고 선택창을 Cancel단추가 닿도록 끌고가서 놓는다. Ctrl+H를 누른다.

③ 폼을 선택하고 Ctrl+L을 누른다.

창문부품들은 수직으로 배치되고 매개는 수직 및 수평으로 최대공간을 모두 채우도록 늘어난다. 단추들은 폼의 전체폭을 모두 채우도록 확장되므로 오히려 더 크게 보인다. 수축자를 리용하여 단추들을 더 작게 만드는것이 좋을것이다. OK단추를 찰칵한 다음 Ctrl+B(배치해체)를 누르고 두개 단추의 크기를 조절하여 단추들의 각 변에 공간이 있게 한다. Spacer도구띠단추를 찰칵하고 OK단추의 왼쪽을 선택한다. 튀여나오기차림표로부터 Horizontal를 선택한다. 이 수축자를 복사하고 두 단추사이에 사본을 배치한다. 수축자를 다시 복사하여 Cancel단추의 오른쪽에 배치한다. (둘째와 셋째 수축자들에 대하여 첫째수축자우를 누르고 Ctrl+C를 누른 다음 Ctrl+V를 누른다. 새로운 수축자를 필요한 위치로 끌고간다.)Ctrl을 누르면서 제일 왼쪽 수축자를 선택하고 선택창을 단추들과 수축자들을 닿도록 끌어다놓고 Ctrl+H를 누른다. 폼을 선택하고 Ctrl+L을 누른다.

이제는 신호와 처리부들을 련결한다. Edit|Connections을 선택하여 View and Edit Connections대화칸을 펼친다.

OK단추의 clicked()신호를 폼의 accept()처리부에 련결하는 새로운 련결을 창조한다. Cancel단추의 clicked()신호와 폼의 reject()처리부를 련결하는 둘째 련결을 창조한다. (2절 6.을 참고.)

특수라지오단추가 선택되면 amount스핀단추를 가능하게 한다. 이번에는 특수라지오단추의 toggled()신호와 amount스핀단추의 setEnabled()처리부에 련결하는 다른 련결을 창조한다.

사용자가 standard 혹은 none라지오단추들을 검사하면 그 량을 각각 설정한다. 대부률단추그룹의 clicked()신호를 새로운 전용처리부 setAmount()(Edit Slots...단추를 찰칵하여 창조한다.)에 련결한다.

폼의 파생클라스를 작성하여 라지오단추가 검사되였는가에 따라서 스핀단추에 량을 설정하려고 한다. 폼을 creditformbase.ui로서 보관한다.(Ctrl+S를 누른다.)

- 시험기구작성

응용프로그람안에서 대화칸을 리용하려고 한다 하더라도 시험기구를 창조하여 독자적으로 개발하고 시험할수 있다. File|New를 선택하여 New File대화칸을 연 다음 C++ Source를 선택하고 OK를 찰칵한다. 펼쳐지는 편집기의 창문에서 다음의 코드를 입력한다.

```
#include <qapplication.h>
#include "creditformbase.h"

int main( int argc, char *argv[] )
{
  QApplication app( argc, argv );

  CreditFormBase creditForm;
  app.setMainWidget( &creditForm );
  creditForm.show();

  return app.exec();
}
```

creditformbase.h를 포함하고 CreditFormBase객체의 실례를 만든다. 일단 파생클라스를 썼다면 머리부를 파생클라스 creditform.h로 변경하고 CreditForm의 실례를 만든다.

이제는 qmake를 사용하여 응용프로그람을 생성한다. 즉 qmake -o Makefile credit.pro. 그리고 make를 실행한다. 폼은 잘 실행되여야 하지만 아직 요구대로 동작하지 않는다.

- 파생클라스의 작성

파생클라스를 위한 머리부와 실현파일을 창조한다. 파생클라스용코드는 최소이다. 머리부파일은 qt/tools/designer/examples/credit/creditform.h이다.

```
#include "creditformbase.h"

class CreditForm : public CreditFormBase
{
  Q_OBJECT
public:
  CreditForm( QWidget* parent = 0, const char* name = 0,
        bool modal = FALSE, WFlags fl = 0 );
  ~CreditForm();
public slots:
  void setAmount();
};
```

Qt Designer에서 처리부 setAmount()를 선언하였다. Q_OBJECT마크로는 신호와 처리부를 사용하는 클라스에서 기본이므로 포함해야 한다.

실현파일 qt/tools/designer/examples/credit/creditform.cpp는 간단하다.

```
#include <qradiobutton.h>
#include <qspinbox.h>
#include "creditform.h"

CreditForm::CreditForm( QWidget* parent, const char* name,
            bool modal, WFlags fl )
  : CreditFormBase( parent, name, modal, fl )
{
  setAmount();
}

CreditForm::~CreditForm() { /* NOOP */ }

void CreditForm::setAmount()
{
  if ( stdRadioButton->isChecked() )
    amountSpinBox->setValue( amountSpinBox->maxValue() / 2 );
```

```
        else if ( noneRadioButton->isChecked() )
            amountSpinBox->setValue( amountSpinBox->minValue() );
    }
```

setAmount()를 호출하여 Qt Designer에서 어느 라지오단추를 검사하였는가에 기초하여 폼이 기동할 때 정확한 량이 표시된다는것을 담보한다. setAmount()에서 standard 혹은 none라지오단추가 선택된다면 량을 설정한다. 사용자가 특수라지오단추를 선택하였다면 량을 변경하는것은 자유이다.

파생클라스를 시험하기 위하여 main.cpp를 변경하여 creditformbase.h가 아니라 creditform.h를 포함하고 creditForm객체의 실례작성을 변경한다.

```
#include <qapplication.h>
#include "creditform.h"

int main( int argc, char *argv[] )
{
    QApplication app( argc, argv );

    CreditForm creditForm;
    app.setMainWidget( &creditForm );
    creditForm.show();

    return app.exec();
}
```

Qt Designer로 creditform.h와 creditform.cpp파일들을 생성하였다면 그것들은 이미 프로젝트파일에 있지만 수동으로 생성하였다면 끝에 두개의 새 행을 추가하여 프로젝트파일을 갱신한다.

```
HEADERS += creditform.h
SOURCES += creditform.cpp
```

폼을 시험하기 위하여 qmake를 실행하여 Makefile을 다시 생성하고 make하고 실행한다.

파생클라스작성실례는 간단하지만 이것은 Qt에서 폼의 파생클라스작성를 반영하며 그것은 간단하다.

2. .ui파일들로부터 동적대화칸의 작성

Qt프로그람은 Qt Designer .ui파일들을 적재하고 .ui파일들에 의하여 표시된 폼들의 실례들을 작성할수 있다. .ui파일은 콤파일되지 않으므로 어떤 C++코드도 포함할수 없다

(실례로 처리부실현). 여기서는 동적대화칸을 적재하는 방법과 동적대화칸의 전용처리부를 실현하는데 사용하는 클라스의 창조방법을 설명한다.

실례폼으로서 소절1에서 창조한 credit폼을 사용한다. 간단히 폼의 실례를 만들고 실행하는것으로 시작하여 전용처리부실현방법을 설명한다.

main.cpp파일을 생성하여 시험기구로 사용하고 수동적으로 프로젝트파일을 창조한다.

1) 프로젝트파일의 작성

프로젝트파일 qt/tools/designer/examples/receiver1/receiver.pro는 다음과 같다.

```
TEMPLATE   = app
CONFIG     += qt  warn_on  release
TARGET      = receiver
SOURCES   += main.cpp
unix:LIBS   += -lqui
win32:LIBS  += $(QTDIR)/lib/qui.lib
FORMS      = mainform.ui
LANGUAGE   = C++
INCLUDEPATH  += $(QTDIR)/tools/designer/uilib
```

creditformbase.ui파일을 실행시에 읽어들이므로 이 파일을 포함하지 않는다. 필요한 기능이 표준Qt서고의 부분이 아니므로 qui서고를 포함해야 한다.

2) main.cpp의 작성

main.cpp는 아주 표준적이다. Qt Designer에서 창조하려고 하는 폼을 기본폼으로서 호출한다. 그다음 이 폼이 적재되고 동적대화칸을 실행한다.

```
#include  <qapplication.h>
#include  "mainform.h"

int  main(  int  argc,  char  *argv[]  )
{
   QApplication  app(  argc,  argv  );

   MainForm  *mainForm  =  new  MainForm;
   app.setMainWidget(  mainForm  );
   mainForm->show();

   return  app.exec();
}
```

MainForm클라스의 새로운 실례를 창조하고 그것을 기본창문부품으로 설정하고 표시하고 app.exec()호출에서 사건순환고리에 들어간다.

3) 기본폼의 작성

- 폼의 설계

① Qt Designer에서 프로젝트파일 receiver.pro를 연다. 동적대화칸을 호출하는데 사용하는 기본창문으로서 대화칸을 창조한다. Ctrl+N을 눌러서 New File대화칸을 열고 OK를 찰칵하여 기정대화칸을 얻는다. 대화칸이름을 MainForm, 제목을 Main Form으로 변경한다. 본문 &Credit Dialog를 가지는 creditPushButton과 본문 &Quit를 가지는 quitPushButton라는 단추를 추가한다. (매개 단추에 대하여 Push Button도구띠단추를 찰칵한 다음 폼을 선택한다. 속성창문에서 속성을 방금 서술한값으로 변경한다.)

② 한쌍의 표식자를 추가하여 동적대화칸에서 사용자가 선택한 설정을 표시할수 있다. Text Label도구띠단추를 찰칵하고 Credit Dialog단추아래의 폼을 선택한다. 표식자의 text를 Credit Rating로 변경한다. Quit단추아래에 다른 본문표식자를 추가한다. name을 ratingTextLabel로, text를 Unrated로 변경한다.

③ 창문부품들을 배치한다. 폼을 선택하고 Ctrl+G(살창배치)를 누른다.

④ 신호-처리부련결을 조절한다. View and Edit Connections대화칸을 펼치고 credit대화칸단추의 clicked()신호를 새로운 creditDialog()(Edit Slots...단추를 찰칵하여 창조한다.)전용처리부와 련결한다. Quit단추의 clicked()신호를 대화칸의 accept()함수와 련결한다.

폼을 mainform.ui라는 이름으로 보관한다. (Ctrl+S를 누르고 파일이름을 입력한다.) 후에 Qt Designer에서 직접 동적대화칸을 적재하고 호출하는 코드를 작성한다.

- 동적대화칸의 적재와 실행

credit대화칸을 호출하는 코드를 추가한다. 그전에 창문부품factory의 머리부파일을 폼에 추가해야 한다. Object Hierarchy에서 Source타브를 선택한다. Includes (in Implementation)를 오른쪽 단추로 찰칵하고 New를 선택한다. <qwidgetfactory.h>라고 입력하고 Enter를 누른다. 동적대화칸에서 스핀단추를 호출해야 하므로 그 머리부파일을 추가해야 한다. Includes (In implimentation)를 오른쪽 단추로 찰칵하고 New를 선택한다. <qspinbox.h>라고 입력한 다음 Enter를 누른다.

기본폼에서 creditDialog()를 호출하는 처리부를 창조한다. Qt Designer에서 이 처리부를 직접 실현하고 그것을 사용하여 동적대화칸을 적재하고 실행한다. 코드는 mainform.ui의 C++실현을 포함하는 qt/tools/designer/examples/receiver1/mainform.ui.h에서 발취하였다.

```
void  MainForm::creditDialog()
{
```

```
    QDialog *creditForm = (QDialog *)
      QWidgetFactory::create( "../credit/creditformbase.ui" );
     // 여기서 동적대화칸을 설정한다.

    if ( creditForm->exec() ) {
      // 사용자가 받아들였으므로 작용한다.
      QSpinBox *amount =
          (QSpinBox *) creditForm->child( "amountSpinBox", "QSpinBox" );
      if ( amount )
        ratingTextLabel->setText( amount->text() );
    }
    delete creditForm;
  }
```

create()함수는 정적 QWidgetFactory함수이다. 이것은 특수화된 .ui파일을 적재하고 .ui파일로부터 생성된 제일 웃준위 QWidget에로의 지적자를 돌려준다. creditformbase.ui파일이 QDialog를 정의한다는것을 알고있으므로 지적자를 QDialog로 강제변환한다. 대화칸을 창조한 다음 그것을 실행한다. 사용자가 OK를 찰칵하면 대화칸은 Accepted를 돌려주고 if명령문의 본체에 들어간다. 우리는 사용자가 선택한 대부량을 알려고 한다. 대화칸상의 child()함수에 창문부품의 이름을 넘기여 호출한다. child()함수는 넘긴 이름을 가지는 창문부품에로의 지적자를 돌려주거나 그런 이름의 창문부품을 발견하지 못하면 0을 돌려준다. 실례에서 child()를 호출하여 amountSpinBox에로의 지적자를 얻는다. 얻은 지적자가 0아니면 비률본문을 대화칸의 스핀칸의 량으로 설정한다. 끝으로 동적대화칸을 삭제한다. 대화칸의 삭제는 그것이 더는 요구되지 않을 때 그 자원을 해방한다.

child()에 창문부품이름을 넘기여 동적대화칸안에 있는 창문부품에로의 호출을 얻었다. 일부 상황에서 창문부품이 무엇을 호출하는지 모른다. null창문부품이름과 클라스이름으로 child()를 호출하여 특정한 클라스의 첫 창문부품을 호출할수 있다(실례로 child(0,"QPushButton"). 이것은 발견한 첫 QPushButton에로의 지적자를 돌려주거나 발견하지 못하면 0을 돌려준다. 주어진 클라스의 모든 창문부품들에로의 지적자를 요구한다면 QObject::queryList()함수에 클라스이름을 넘기여 호출한다. 그러면 주어진 클라스로부터 파생되는 대화칸안의 각 객체를 가리키는 QObjectList지적자를 돌려준다(QObject문서를 참고).

- 동적대화칸용처리부의 실현

한가지 언급하지 않은 문제가 있다. 동적대화칸은 setAmount()처리부를 실현하지 않

았으므로 원시적인 신용대화칸의 동작을 가지지 못한다. QObject파생클라스를 창조하여 동적대화칸용처리부들을 실현한다. 그다음 파생클라스의 실례를 창조하고 그 지적자를 동적대화칸의 신호들을 파생클라스에서 실현된 처리부들에 련결하는 QWidgetFactory::create()함수에로 넘긴다.

QObject파생클라스를 창조하고 creditDialog()를 변경하여 QWidgetFactory::create()함수에 넘길수 있는 파생클라스의 실례를 창조해야 한다. 여기에 mainform.ui의 처리부용코드를 포함하는 qt/tools/designer/examples/receiver2/mainform.ui.h파일로부터 수정된 creditDialog()함수가 있다.

```
void MainForm::creditDialog()
{
    Receiver *receiver = new Receiver;
    QDialog *creditForm = (QDialog *)
    QWidgetFactory::create( "../credit/creditformbase.ui", receiver );
    receiver->setParent( creditForm );

    // Set up the dynamic dialog here

    if ( creditForm->exec() ) {
        // The user accepted, act accordingly
        QSpinBox *amount = (QSpinBox *) creditForm->child(
                           "amountSpinBox", "QSpinBox" );
        if ( amount )
            ratingTextLabel->setText( amount->text() );
    }

    delete receiver;
    delete creditForm;
}
```

Receiver파생클라스의 새로운 실례를 창조한다. (이 클라스의 코드를 간단히 쓴다.) 그다음 QWidgetFactory::create()를 사용하여 QDialog를 창조한다. 이 호출은 create()함수가 자동적으로 신호와 처리부들을 설정할수 있도록 하기 위하여 파생클라스객체에서 넘기므로 이전의 실례와 차이난다. 우리의 처리부는 동적폼의 창문부품들을 호출해야 하므로 setParent()함수를 통하여 수신자객체에 폼에로의 지적자를 넘긴다. 함수의 나머지는 수신자객체를 삭제하는것을 제외하면 앞에서와 같다.

기본폼에서 Receiver파생클라스를 사용하므로 그 머리부파일을 포함해야 한다. Object Explorer의 Members타브에서 Includes (in Implmentation)를 오른쪽 단추로 찰칵하고 New를 선택한다. receiver.h라고 입력하고 Enter를 누른다.

이제는 Receiver파생클라스의 실현을 고찰한다. 코드는 qt/tools/designer/examples/receiver2/receiver.h와 대응하는 receiver.cpp파일에서 발취한다. 머리부파일로부터 시작한다.

```
#include <qobject.h>
#include <qdialog.h>

class Receiver : public QObject
{
  Q_OBJECT
public:
  void setParent( QDialog *parent );
public slots:
  void setAmount();
private:
  QDialog *p;
};
```

클라스는 QObject파생클라스이고 신호와 처리부들을 사용하므로 Q_OBJECT마크로를 포함한다. 비공개 QDialog지적자는 물론 실현하려는 함수와 setAmount()를 선언한다.

이 실현은 사용하는 클라스들의 머리부파일들을 요구한다.

```
#include <qradiobutton.h>
#include <qspinbox.h>
#include "receiver.h"
```

receiver.cpp에서 각 함수의 실현을 론의한다.

```
void Receiver::setParent( QDialog *parent )
{
  p = parent;
  setAmount();
}
```

setParent()함수는 동적대화칸에로의 지적자를 비공개지적자에 대입한다. QWidgetFactory::create()를 호출하기전에 Receiver객체를 구성해야 하는데 구성자호출에서 이것을 수행할수 없으므로 Receiver객체를 create()함수에 넘겨야 한다. 일단 create()

함수를 호출한 다음 setParent()를 통하여 Receiver클라스에 넘길수 있는 동적대화칸에로의 지적자를 가진다. 이 실례의 파생클라스판에서 구성자에서 setAmount()를 호출하지만 setAmount()의 실현이 구성시에 유효하지 않은 동적대화칸에 의존하므로 여기서 그것을 수행할수 없다. 이리하여 setParent()함수에서 setAmount()를 호출한다.

```
void Receiver::setAmount()
{
  QSpinBox *amount = (QSpinBox*)p->child( "amountSpinBox", "QSpinBox");

  QRadioButton *radio =
    (QRadioButton *) p->child( "stdRadioButton", "QRadioButton" );
  if ( radio && radio->isChecked() ) {
    if ( amount )
      amount->setValue( amount->maxValue() / 2 );
    return;
  }

  radio = (QRadioButton *) p->child( "noneRadioButton", "QRadioButton" );
  if ( radio && radio->isChecked() )
    if ( amount )
      amount->setValue( amount->minValue() );
}
```

량스핀단추를 갱신할수도 있으므로 그것에로의 지적자를 얻어야 한다. setParent()호출에서 할당된 동적대화칸을 가리키는 지적자 p에 대하여 child()를 호출한다. 그 결과의 지적자를 정확한 형에로 강제변환하여 그 형과 련관된 함수들을 호출할수 있다. 실례에서 child()를 호출하여 량스핀칸에로의 지적자를 얻은 다음 다시 child()을 호출하여 stdRadioButton에로의 지적자를 얻는다. 라지오단추에로의 지적자를 얻어서 주어지는 량을 설정하는 단추가 검사되면 량스핀단추에로의 지적자를 가진다. 이 라지오단추가 선택되였으면 완료하고 되돌아간다. stdRadioButton가 선택되지 않으면 noneRadioButton에로의 지적자를 얻고 이 단추가 선택되였으면 량을 설정한다. specialRadioButton가 선택되였으면 사용자가 그 선택값을 자유로 입구하므로 아무것도 하지 않는다.

- 대화칸의 콤파일과 동적적재

《콤파일된》.ui파일과 동적적재된 .ui파일의 사용에서 차이는 다음과 같다.

동적대화칸은 .ui파일에 C++코드를 가질수 없으며 전용처리부는 QObject파생클라스

를 통하여 실현되여야 한다. 콤파일된 대화칸은 .ui파일 혹은 파생클라스에 코드를 포함할수 있다.

동적대화칸은 .ui파일을 읽어야 하고 QWidget실례가 .ui파일의 구문나무에 기초하여 실례화되기때문에 적재속도가 뜨다. 콤파일된 코드는 어떤 파일도 읽어들이지 않고 구문해석도 필요하지 않으므로 훨씬 더 빨리 적재된다. 사용자는 그 차이가 1s에 불과하므로 모른다.

동적대화칸은 코드에 영향이 없는한 코드와 독자적으로 .ui파일을 변경할수 있게 한다. 이것은 폼의 표시를 변경할수 있다는것을 의미한다. 실례로 창문부품이동과 그것들의 배치. 콤파일된 대화칸을 변경하려고 한다면 .ui파일을 변경하고 다시 콤파일해야 한다. 응용프로그람을 건설하고있는데 손님에게 사용자대면부의 모양을 전용화할수 있게 하려면 그에게 Qt Designer의 사본을 주고 동적대화칸을 사용할수 있다.

제6절. 사용자정의창문부품의 작성

사용자정의창문부품은 코드로 창조된다. 그것은 현존창문부품들의 조합으로 이루어지지만 추가적인 기능과 처리부, 신호를 가지거나 또는 새로 만들수도 있으며 두가지를 결합할수도 있다.

Qt Designer는 사용자정의창문부품들을 결합하는 두가지 기구를 제공한다. 즉

① 원시적인 방법은 대화칸칸을 완성하는것보다 좀 더 복잡하다. 이 방법으로 결합한 창문부품들은 Qt Designer에서 폼에 추가되였을 때 지어는 미리보기방식에서도 평면픽스매프로서 나타난다. 그것들은 실행시에 진짜 폼에만 나타난다. 소절1에서 원시적인 수법으로 사용자정의창문부품들을 창조하는 방법을 설명한다.

② 새로운 수법은 플라그인에 창문부품의 매몰을 포함한다. 플라그인을 통하여 삽입되는 창문부품들은 Qt Designer에서 폼을 배치할 때와 미리보기방식에서 폼에 나타난다. 이 수법은 원시수법보다 더 강력하고 유연하고 소절2에서 설명한다.

1. 간단한 사용자정의창문부품

사용자정의창문부품을 창조하는데 2개의 단계가 있다. 첫째로 창문부품을 정의하는 클라스를 창조해야 하며 둘째로 창문부품을 Qt Designer에 삽입해야 한다. 창문부품창조는 단순한 사용자정의창문부품을 창조하거나 플라그인을 창조하여 수행되여야 하지만 단순한 사용자정의창문부품에 대하여 Qt Designer에로의 삽입은 아주 간단하다.

4개의 단추 즉 rewind와 play, next, stop로 이루어지는 VCR형 창문부품을 창조한다. 창문부품은 어느 단추를 눌렀는가에 따라서 신호를 발생한다.

1) 사용자정의창문부품코드작성

사용자정의창문부품은 특별한 결합으로 모두 배치된 하나이상의 표준창문부품들로 이루어지거나 무로부터 쓸수 있다. 사용자정의창문부품에 기초하여 폼에 QPushButton 창문부품들을 결합할수 있다.

우선 머리부파일 qt/tools/designer/examples/vcr/vcr.h를 고찰하자.

```
#include <qwidget.h>

class Vcr : public QWidget
{
    Q_OBJECT
public:
    Vcr( QWidget *parent = 0, const char *name = 0 );
    ~Vcr() {}
signals:
    void rewind();
    void play();
    void next();
    void stop();
};
```

QWidget로부터 사용자정의창문부품을 파생하므로 qwidget.h를 포함한다. 창문부품을 창조하는 구성자와 창문부품에 발송하는 4개의 신호를 선언한다. 신호를 사용하므로 Q_OBJECT마크로를 포함한다.

알아두기: 신호를 리용하므로 역시 Q_OBJECT마크로를 포함해야 한다. 이 마크로는 또한 클라스에 대한 정보는 메타객체체계를 거쳐서 사용된다는것과 Qt Designer가 창문부품에 대한 정보를 현시한다는것을 담보한다.

실현은 간단하다. 우리가 실현하는 유일한 함수는 구성자이다. 파일의 나머지는 include문과 매몰된 .xpm화상들로 구성된다.

```
Vcr::Vcr( QWidget *parent, const char *name ) : QWidget( parent, name )
{
    QHBoxLayout *layout = new QHBoxLayout( this );
    layout->setMargin( 0 );

    QPushButton *rewind = new QPushButton( QPixmap( rewind_xpm ), 0,
                                        this, "vcr_rewind" );
    layout->addWidget( rewind );
```

단추들을 배치할 QHBoxLayout를 창조한다. 코드에서는 오직 rewind단추만 보여준다. 다른 단추들은 단추의 이름과 픽스매프, 신호를 제외하하고는 모두 등가하다. 매개 단추에 대하여 적당히 매몰된 픽스매프를 넘기여 QPushButton구성자를 호출한다. 그다음 단추를 배치에 추가한다. 끝으로 단추의 clicked()신호를 적당한 신호에 련결한다. clicked()신호는 창문부품에 고유하지 않으므로 창문부품의 사용을 반영하는 신호를 발생하려고 한다. rewind(), play(), 기타 신호들은 창문부품에서 의미있으므로 매개 단추의 clicked()신호를 적당한 창문부품고유신호에 전달해야 한다.

실현은 끝났지만 창문부품을 콤파일하고 실행을 확인하기 위하여 간단한 시험기구를 창조한다. 시험기구는 2개의 파일 즉 .pro프로젝트파일과 main.cpp파일을 요구한다. qt/tools/designer/examples/vcr/vcr.pro프로젝트파일은 다음과 같다.

```
TEMPLATE = app
LANGUAGE = C++
TARGET   = vcr

CONFIG  += qt warn_on release
SOURCES += vcr.cpp main.cpp
HEADERS += vcr.h
DBFILE   = vcr.db
```

qt/tools/designer/examples/vcr/main.cpp파일의 기본은 다음과 같다.

```
#include <qapplication.h>
#include "vcr.h"

int main( int argc, char ** argv )
{
   QApplication app( argc, argv );
   Vcr *vcr = new Vcr;
   vcr->show();
   return app.exec();
}
```

사용자정의창문부품이 만족스럽게 콤파일되고 실행된다면 그것을 Qt Designer에 포함할수 있다.

기초클라스형판에서 용기사용자정의창문부품의 창조를 설명한다.

2) Qt Designer에 사용자정의창문부품의 추가

Tools|Custom|Edit Custom Widgets를 선택하여 Edit Custom Widgets대화칸

을 연다.

① New Widget를 선택하고 새 창문부품을 추가할 준비를 한다.

② 클라스이름을 MyCustomWidget로부터 Vcr로 변경한다.

③ Headerfile행편집의 오른쪽에 있는 생략(...)단추를 찰칵하여 파일열기대화칸을 연다. vcr.h를 찾아서 선택하고 Open을 찰칵한다. 그것이 머리부파일로서 나타난다.

④ 도구띠에서 창문부품을 식별하는데 사용하는 픽스매프를 가지고있다면 Pixmap속성의 오른쪽에 있는 생략단추를 찰칵한다. (생략단추는 Properties목록의 Value부분을 선택할 때 pixmap나 iconSet속성에 의하여 나타난다.)

실례에는 이 목적에 사용하는 파일 qt/tools/designer/examples/vcr/play.xpm이 있다.

⑤ 창문부품의 최소감별크기를 알고있으므로 이 값들을 Size Hint스핀칸에 넣는다. 폭을 80(왼쪽스핀칸에서), 높이를 20 (오른쪽스핀칸에서)으로 입력한다.

완성해야 할 나머지 항목들은 우리가 창조하는 창문부품의 특성에 의존한다. 실례로 자기의창문부품을 Container Widget검사칸에서 검사한 창문부품들을 포함하는데 사용할수 있다. Vcr실례의 경우에 추가하려는 항목들은 그 신호들이다.

Signals타브를 선택한다. New Signal단추를 찰칵하고 신호이름을 rewind()라고 입력한다. 다시 New Signal을 선택하고 이번에는 play()라고 입력한다. 같은 방법으로 next()와 stop()신호들을 추가한다.

우리의 실례는 처리부나 속성을 가지지 않으므로 Close를 선택할수 있다. 새 그림기호가 새 창문부품을 표시하는 Qt Designer의 도구띠에 나타난다. 새 폼을 창조하면 Vcr창문부품들을 추가하고 자기 처리부에 Vcr의 신호를 련결할수 있다.

자체의 처리부와 속성을 가지는 사용자정의창문부품의 삽입은 신호추가와 류사한 방법으로 진행된다. 필요한 모든 정보는 사용자정의창문부품의 머리부파일에 있다.

2. 플라그인형식의 사용자정의창문부품의 작성

여기서는 사용자정의창문부품을 쓰는 방법과 플라그인으로 사용자정의창문부품을 매몰하는 방법을 보여준다. 플라그인을 목적으로 하는 창문부품을 창조할 때 고려해야 할 제한은 없다. 경험있는 Qt프로그람작성자라면 이 항을 뛰여넘어 플라그인의 작성으로 직접 넘어갈수 있다.

창문부품을 전용화하는데 플라그인수법을 사용한다면 플라그인은 콤파일시에 Qt Designer뿐아니라 uic에도 유효하여야 한다는것을 알아야 한다.

1) 사용자정의창문부품의 작성

보통 사용자정의창문부품은 다른 창문부품의 특수화(파생클라스) 또는 함께 작업하는 창문부품들의 결합 혹은 이 두가지 수법들의 혼합물이다. 특수한 환경에서 창문부품들의 집합만 요구되면 그것들을 창조하는것은 가장 쉽다. 그것들을 그룹으로 선택하고 Qt Designer에서 요구될 때 그것들을 복사하여 배치한다. 일반적으로 사용자정의창문부

품은 현존창문부품이나 창문부품그룹에 새 기능을 추가하려고 할 때 창조한다.

플라그인으로서 사용자정의창문부품을 창조할 때 고려해야 할 문제가 두가지있다.

① Qt의 속성체계를 사용하면 속성편집기를 통하여 창문부품환경을 구성하는 직접적인 수단을 Qt Designer사용자에게 제공한다.

② 자기 창문부품의 공개set함수들을 공개처리부로 만들어서 Qt Designer에서 창문부품와의 신호-처리부련결을 처리할수 있다.

이 절에서 간단하지만 쓸모있는 창문부품 FileChooser를 만든다. 이것은 후에 플라그인으로서 Qt Designer에서 사용할수 있게 한다. 실천에서 대부분의 사용자정의창문부품들은 창문부품들을 구성하는것보다도 오히려 기능을 추가하기 위하여 창조되므로 이 수법을 반영하기 위하여 Qt Designer에서 사용하는것보다 코드로 창문부품들을 창조한다. FileChooser는 QLineEdit와 QPushButton으로 이루어진다. QLineEdit는 파일이나 등록부이름을 보관하는데 사용되며 QPushButton은 사용자가 파일이나 등록부를 선택할수 있는 파일대화칸을 펼치는데 사용된다.



그림 1-65. FileChooser사용자정의창문부품

이제는 자체로 이 사용자정의창문부품을 창조할수 있다. 자체의 창문부품판을 만들거나 플라그인으로 하려는 다른 창문부품이 있다면 플라그인의 작성으로 넘어갈수 있다.

- 창문부품대면부코드의 작성

창문부품의 머리부파일 qt/tools/designer/examples/filechooser/widget/filechooser.h를 통하여 한걸음씩 작업한다.

```
#include <qwidget.h>
#include <qwidgetplugin.h>


class QLineEdit;
class QPushButton;
```

우리의 창문부품은 QWidget로부터 파생되므로 qwidget.h머리부파일을 포함한다. 또한 창문부품이 건설되는 2개의 클라스들을 앞방향선언한다.

이것은 신호나 처리부를 선언하는 클라스들을 요구하므로 Q_OBJECT마크로를 포함한다. Q_ENUMS선언은 Mode렬거를 등록하는데 쓰인다. 우리의 창문부품은 2개의 속성 즉 사용자가 파일을 선택하는가 등록부를 선택하는가를 보관하는 mode와 선택한 파일이나 등록부를 보관하는 fileName을 가진다.

```
class QT_WIDGET_PLUGIN_EXPORT FileChooser : public QWidget
{
```

```
    Q_OBJECT
    Q_ENUMS( Mode )
    Q_PROPERTY( Mode mode READ mode WRITE setMode )
    Q_PROPERTY( QString fileName READ fileName WRITE setFileName )

public:
    FileChooser( QWidget *parent = 0, const char *name = 0);

    enum Mode { File, Directory };

    QString fileName() const;
    Mode mode() const;
```

구성자는 창문부품에 대하여 표준적인 방법으로 선언된다. 2개의 공개함수 즉 파일이름을 돌려주는 fileName()과 방식을 돌려주는 mode()를 선언한다.

```
public slots:
    void setFileName( const QString &fn );
    void setMode( Mode m );

signals:
    void fileNameChanged( const QString & );

private slots:
    void chooseFile();
```

2개의 set함수를 공개처리부로 선언한다. setFileName()과 setMode()는 파일이름과 방식을 각각 설정한다. 하나의 신호 fileNameChanged()를 선언한다. 비공개처리부 chooseFile()는 단추를 찰칵할 때 창문부품자체에 의하여 호출된다.

```
private:
    QLineEdit *lineEdit;
    QPushButton *button;
    Mode md;

};
```

Mode변수는 물론 QLineEdit와 QPushButton에로의 지적자는 비공개자료로 보관된다.

- 실현코드작성

qt/tools/designer/examples/filechooser/widget/filechooser.cpp에 있는 실현을 통하여 한걸음씩 작업하자.

```
FileChooser::FileChooser( QWidget *parent, const char *name )
    : QWidget( parent, name ), md( File )
{
```

구성자는 그 상위클라스 QWidget에로 parent와 name을 넘기며 또한 비공개방식자료 md를 File방식으로 초기화한다.

```
    QHBoxLayout *layout = new QHBoxLayout( this );
    layout->setMargin( 0 );

    lineEdit = new QLineEdit( this, "filechooser_lineedit" );
    layout->addWidget( lineEdit );
```

수평칸배치(QHBoxLayout)를 창조하는것으로 시작하여 하나의 QLineEdit를 추가하고 거기에 하나의 QPushButton을 추가한다.

```
    connect( lineEdit, SIGNAL( textChanged( const QString & ) ),
             this, SIGNAL( fileNameChanged( const QString & ) ) );

    button = new QPushButton( "...", this, "filechooser_button" );
    button->setFixedWidth( button->fontMetrics().width( " ... " ) );
    layout->addWidget( button );

    connect( button, SIGNAL( clicked() ), this, SLOT( chooseFile() ) );
```

lineEdit의 textChanged()신호를 사용자정의창문부품의 fileNameChanged()신호에 련결한다. 이것은 사용자가 QLineEdit의 본문을 변경한다면 사용자정의창문부품의 자기 신호를 통하여 이 사실이 전달된다는것을 담보한다. 단추의 clicked()신호는 사용자에게 적당한 대화칸을 펼치여 파일이나 등록부를 선택하는 사용자정의창문부품의 chooseFile()처리부에 련결한다.

```
    setFocusProxy( lineEdit );
}
```

lineEdit를 사용자정의창문부품용의 초점대리로서 설정한다. 이것은 창문부품에 초점이 주어질 때 초점이 실제로 lineEdit로 간다는것을 의미한다.

```
void FileChooser::setFileName( const QString &fn )
{
```

```
   lineEdit->setText( fn );
}

QString FileChooser::fileName() const
{
   return lineEdit->text();
}
```

setFileName()함수는 QLineEdit에서 파일이름을 설정하고 fileName()함수는 QLineEdit로부터 파일이름을 돌려준다. 마찬가지로 setMode()와 mode()함수들을 설정하고 주어진 방식을 돌려준다.

```
void FileChooser::chooseFile()
{
   QString fn;
   if ( mode() == File )
      fn = QFileDialog::getOpenFileName( lineEdit->text(), QString::null,this);
   else
      fn = QFileDialog::getExistingDirectory( lineEdit->text(), this );

   if ( !fn.isEmpty() ) {
      lineEdit->setText( fn );
      emit fileNameChanged( fn );
   }
}
```

chooseFile()이 호출될 때 방식에 따라 사용자에게 파일 혹은 등록부대화칸을 표시한다. 사용자가 파일이나 등록부를 선택하면 QLineEdit는 선택된 파일이나 등록부로 갱신되고 fileNameChanged()신호가 발생된다.

비록 2개의 파일이 FileChooser창문부품의 실현을 완성한다 할지라도 시험기구를 작성하여 창문부품을 플라그인으로 넣기전에 기대한대로 동작하는가를 검사하는것이 좋다.

- 실현의 시험

우리의 사용자정의창문부품을 실행하게 하는 기본시험기구를 제공한다. 시험기구는 두개의 파일 즉 FileChooser를 포함하는 main.cpp와 Makefile을 포함하는 .pro파일을 요구한다. 아래에 qt/tools/designer/examples/filechooser/widget/main.cpp가 있다.

```
#include <qapplication.h>
#include "filechooser.h"
```

```
int main( int argc, char ** argv )
{
   QApplication a( argc, argv );
   FileChooser *fc = new FileChooser;
   fc->show();
   return a.exec();
}
```

또한 아래에 qt/tools/designer/examples/filechooser/widget/filechooser.pro가 있다.

```
TEMPLATE = app
LANGUAGE = C++
TARGET    = filechooser

SOURCES += filechooser.cpp main.cpp
HEADERS += filechooser.h
CONFIG   += qt warn_on release
DBFILE   = filechooser.db
DEFINES += FILECHOOSER_IS_WIDGET
```

qmake를 사용하여 다음과 같이 makefile을 창조한다. 즉 qmake -o Makefile filechooser.pro이다. 그다음 시험기구를 make하고 실행하여 새 창문부품을 시험한다. 사용자정의창문부품이 튼튼하고 필요한 동작을 한다는것이 확인되면 그것을 플라그인으로 매몰할수 있다.

2) 플라그인의 작성

Qt Plugins은 Qt응용프로그람에 자체로 포함한 쏘프트웨어부분품들을 제공하는데 사용될수 있다. Qt는 현재 5가지 종류의 플라그인 즉 codecs와 image formats, database drivers, styles, custom widgets창문부품의 창조를 지원한다. 여기서는 filechooser사용자정의창문부품을 Qt Designer사용자정의창문부품플라그인으로 변환하는 방법을 설명한다.

Qt Designer사용자정의창문부품플라그인은 항상 QwidgetPlugin으로부터 파생된다. 필요한 코드량은 최소이다.

자기의 플라그인을 만들기 위하여서는 실례 plugin.h과 plugin.cpp파일들을 복사하여 CustomWidgetPlugin을 자기의 창문부품플라그인실현클라스에 사용하려는 이름으로 변경하는것부터 시작하는것이 가장 편리하다. 클라스이름변경후에 다른 변경이 요구되지 않는다 하더라도 머리부파일에 대한 소개를 다음에 제공한다. 실현파일은 간단한 변경

대체로는 클라스이름변경을 요구한다. 매개 함수를 차례로 개괄하고 무엇을 해야 하는가를 설명한다.

- CustomWidgetPlugin실현

우리의 플라그인클라스를 사용하여 사용자정의창문부품들을 은페하려고 하므로 머리부파일을 plugin.h, 플라그인클라스를 CustomWidgetPlugin으로 한다. 전체머리부파일을 제시하여 필요한 실현의 리용범위를 암시한다. 대부분의 함수들은 여러행의 코드를 요구한다.

```
// qt/tools/designer/examples/filechooser/plugin/plugin.h로부터 발취
#include <qwidgetplugin.h>

class CustomWidgetPlugin : public QWidgetPlugin
{
public:
    CustomWidgetPlugin();

    QStringList keys() const;
    QWidget* create( const QString &classname, QWidget* parent = 0,
                     const char* name = 0 );
    QString group( const QString& ) const;
    QIconSet iconSet( const QString& ) const;
    QString includeFile( const QString& ) const;
    QString toolTip( const QString& ) const;
    QString whatsThis( const QString& ) const;
    bool isContainer( const QString& ) const;
};
```

- QWidgetPlugin함수

우리의 플라그인 .cpp파일을 복사하고 CustomWidgetPlugin의 모든 실현을 자기 창문부품플라그인실현에 사용하려는 이름으로 변경하여 자체의 플라그인 .cpp파일을 창조하자. 다른 변경은 전용조종요소 FileChooser의 이름을 자기 전용조종요소의 이름으로 단순히 교체하는것이다. 자기의 플라그인실현에 하나이상의 전용조종요소가 있다면 여분의 else if절을 추가할수 있다.

이제 구성자를 고찰하자.

```
CustomWidgetPlugin::CustomWidgetPlugin()
{
```

```
}
```

구성자는 아무것도 하지 않는다. 자체의 창문부품플라그인실현에 사용하려는 클라스이름으로 우리의것을 단순히 복사한다.

해체자는 필요없다.

- keys함수

```
QStringList CustomWidgetPlugin::keys() const
{
    QStringList list;
    list << "FileChooser";
    return list;
}
```

플라그인실현에 은페하려는 매개 창문부품클라스에 대하여 클라스를 식별할수 있는 건을 제공해야 한다. 이 건은 자기 클라스의 이름이여야 하므로 실례에서는 단일건 FileChooser을 추가한다.

- create()함수

```
QWidget* CustomWidgetPlugin::create( const QString &key,
                                     QWidget* parent, const char* name )
{
    if ( key == "FileChooser" )
        return new FileChooser( parent, name );
    return 0;
}
```

이 함수에서는 요구되는 클라스의 실례를 창조하고 새로 창조한 창문부품에로의 QWidget지적자를 돌려준다. 클라스이름과 특성이름을 변경하여 이 함수를 복사하고 방금 여기서 수행한것처럼 자기 창문부품의 실례를 창조한다.

- includeFile()함수

```
QString CustomWidgetPlugin::includeFile( const QString& feature ) const
{
    if ( feature == "FileChooser" )
        return "filechooser.h";
    return QString::null;
}
```

이 함수는 사용자정의창문부품용의 머리부파일이름을 돌려준다. 이 함수를 클라스이름과 건, 머리부파일이름을 자기 사용자정의창문부품에 맞게 변경하여 복사한다.

- group()와 iconSet(), toolTip(), whatsThis()함수들

```
QString CustomWidgetPlugin::group( const QString& feature ) const
{
  if ( feature == "FileChooser" )
    return "Input";
  return QString::null;
}

QIconSet CustomWidgetPlugin::iconSet( const QString& ) const
{
  return QIconSet( QPixmap( filechooser_pixmap ) );
}

QString CustomWidgetPlugin::includeFile( const QString& feature ) const
{
  if ( feature == "FileChooser" )
    return "filechooser.h";
  return QString::null;
}

QString CustomWidgetPlugin::toolTip( const QString& feature ) const
{
  if ( feature == "FileChooser" )
    return "File Chooser Widget";
  return QString::null;
}

QString CustomWidgetPlugin::whatsThis( const QString& feature ) const
{
  if ( feature == "FileChooser" )
    return "A widget to choose a file or directory";
  return QString::null;
}
```

group()함수를 사용하여 어느 Qt Designer도구띠그룹이 이 사용자정의창문부품을

포함하는가를 식별한다. 사용하지 않는 이름을 리용하면 Qt Designer는 주어진 이름을 가지는 새 도구띠그룹을 창조한다. 이 함수를 클라스이름, 건과 그룹이름을 자기의 창문부품플라그인실현에 적합하게 변경한다.

iconSet()함수는 도구띠에서 사용자정의창문부품을 표시하는데 사용하는 픽스매프를 돌려준다. toolTip()함수는 도구암시본문을 돌려주고 whatsThis()함수는 Whats This본문을 돌려준다. 이 함수들을 각각 클라스이름과 건, 돌려주는 문자렬을 자기의 창문부품플라그인실현에 적합하게 변경하여 복사한다.

- isContainer()함수

```
bool CustomWidgetPlugin::isContainer( const QString& ) const
{
    return FALSE;
}
```

이 함수에서 클라스이름을 자기의 창문부품플라그인실현에 적합하게 변경하여 복사한다. 자기의 사용자정의창문부품이 다른 창문부품(가령 QFrame)을 포함할수 있다면 TRUE를 돌려주어야 한다. 다른 창문부품(가령 QPushButton)을 포함하면 안되는 경우에 FALSE를 돌려준다.

- Q_EXPORT_PLUGIN macro마크로

```
Q_EXPORT_PLUGIN( CustomWidgetPlugin )
```

이 마크로는 모듈을 플라그인으로서 식별한다. 다른 모든 코드는 단지 련관된 대면부를 실현한다. 즉 유효하게 하려는 클라스들을 포함한다.

이 마크로는 자기의 플라그인에 한번 나타나야 한다. 그것은 자기의플라그인의 클라스이름으로 변경된 클라스이름으로 복사되여야 한다.

창문부품플라그인실현에 포함한 매개 창문부품은 플라그인실현이 제공하는 클라스로 된다. 플라그인실현에 포함할수 있는 클라스의 수에는 제한이 없다.

- 프로젝트파일

플라그인용 프로젝트파일은 응용프로그람의 프로젝트파일과 좀 다르지만 대부분의 경우에 HEADERS와 SOURCES행만 변경하여 프로젝트파일을 사용할수 있다.

```
TEMPLATE = lib
LANGUAGE = C++
TARGET    = filechooser

SOURCES   += plugin.cpp ../widget/filechooser.cpp
HEADERS   += plugin.h ../widget/filechooser.h
DESTDIR    = ../../../../../plugins/designer
```

```
target.path=$$plugins.path

INSTALLS    += target
CONFIG       += qt  warn_on  release  plugin
INCLUDEPATH  += $$QT_SOURCE_TREE/tools/designer/interfaces
DBFILE        = plugin.db
```

qt/tools/designer/examples/filechooser/plugin/plugin.pro

HEADERS행을 변경하여 자기의플라그인의 머리부파일과 자기의창문부품용 머리부파일을 표시한다. SOURCES행도 등가하게 변경한다. qmake로 Makefile을 창조하고 프로젝트를 make하면 플라그인이 창조되고 Qt Designer가 발견할수 있는 등록부에 배치된다. 다음번에 Qt Designer를 실행할 때 새로운 플라그인을 발견하고 자동적으로 적재하며 지정된 도구띠에 그림기호가 표시된다.

- 창문부품플라그인의 리용

플라그인이 일단 콤파일되였으면 그것은 자동적으로 발견되고 다음번에 Qt Designer를 실행할 때 Qt Designer에 의하여 적재된다. 자기의 사용자정의창문부품을 다른 창문부품처럼 사용한다.

프로젝트의 다른 곳에서 그 플라그인을 사용하려면 프로젝트에 적당한 행을 추가하여 거기에 련결할수 있다. 실례로 프로젝트의 .pro파일에 다음과 같이 추가한다.

```
LIBS += filechooser.lib
```

응용프로그람을 배포하려고 한다면 실행파일과 함께 번역된 플라그인을 포함한다. Qt설치등록부의 plugins/widgets보조등록부에 플라그인을 설치한다. 표준플라그인경로를 사용한다면 플라그인에 사용하려는 경로를 설치과정에 결정하고 응용프로그람이 실행시에 읽어들이도록 그 경로를 (가령 QSettings를 사용하여) 보관한다. 그다음 응용프로그람은 이 경로를 가지고 QApplication::addLibraryPath()를 호출할수 있으며 자기의 플라그인을 응용프로그람에 사용할수 있다. 경로의 마지막 부분 즉 styles, widgets 등은 변경할수 없다.

- 플라그인과 스레드응용프로그람

스레드화된 Qt서고에서 사용하려는 플라그인(어차피 플라그인자체가 스레드를 사용한다.)을 건설하려고 한다면 스레드환경을 사용해야 한다. 특히 스레드화된 Qt서고를 사용해야 하며 그 서고를 가지고 Qt Designer를 건설해야 한다. 자기 플라그인용 .pro파일에는 다음의 행을 포함해야 한다.

```
CONFIG += thread
```

응용프로그람에 표준Qt서고와 스레드화된 Qt서고를 섞지 말아야 한다. 응용프로그

람이 스레드화된 Qt서고를 사용한다면 표준Qt서고와 련결하지 말아야 한다. 표준Qt서고를 동적으로 적재하지 말아야 하고 또한 다른 서고 실례로 표준Qt서고에 의존하는 플라그인을 동적으로 적재하지 말아야 한다. 일부 체계에서 스레드화서고와 비스레드화서고 또는 플라그인의 혼합은 Qt서고에서 사용된 정적자료를 못쓰게 만든다.

제7절. 자료기지응용프로그람의 작성

이 절에서는 Qt Designer안으로부터 Qt의 자료인식(data-aware)창문부품을 사용하는 방법을 보여준다. 여기서는 QDataTable(표)와 QDataBrowser(폼)에서 INSERT와 UPDATE, DELETE를 보여준다. 또한 주-세부관계와 구멍내기(drilldown)의 코드작성방법을 보여준다. 여기서는 간단한 외부열쇠조종수법을 제시하며 더 복잡한 수법은 직결SQL모듈문서에서 보여준다.

이 창문부품들을 사용하여 실례를 실행하거나 자체의 응용프로그람을 창조하려고 한다면 SQL자료기지와 자료기지에 련결할수 있는 Qt자료기지 구동프로그람을 호출해야 한다. Qt가 지원하는 구동프로그람은 QODBC3 (Open Database Connectivity), QOCI8 (Oracle), QPSQL7 (PostgreSQL 6과 7) 그리고 QMYSQL3 (MySQL)이다.

Qt자료인식창문부품들을 사용하여 SQL을 쓰려고 하지 않고 SQL자료기지에서 자료를 열람 및 편집할수 있더라도 SQL에 대한 기본리해가 강하게 권고된다. SELECT와 INSERT, UPDATE, DELETE문을 알고있는것으로 가정한다. 또한 주열쇠와 외부열쇠의 표준개념을 알고있는것으로 가정하자. SQL자료기지를 론하는 표준서적은 문헌[2]를 참고하시오.

다음에 book자료기지응용프로그람의 창조를 설명한다. 응용프로그람은 in-place레코드편집을 포함하는 QDataTable들의 사용방법과 QDataTable들사이의 주-세부관계를 설정하는 방법을 보여준다. 또한 QDataTable로부터 다른 창문부품 례를 들면 QDataBrowser 혹은 QDataView를 펼치는 방법과 QDataBrowser에서 레코드편집을 처리하는 방법을 설명한다. 파생클라스작성이 항상 더 고급한 조종에 사용할수 있다하더라도 Qt Designer에서 직접 클라스들로부터 대량적인 기능을 사용할수 있다. Book실레를 건설하려고 한다면 자료기지의 실레스키마를 창조해야 한다.

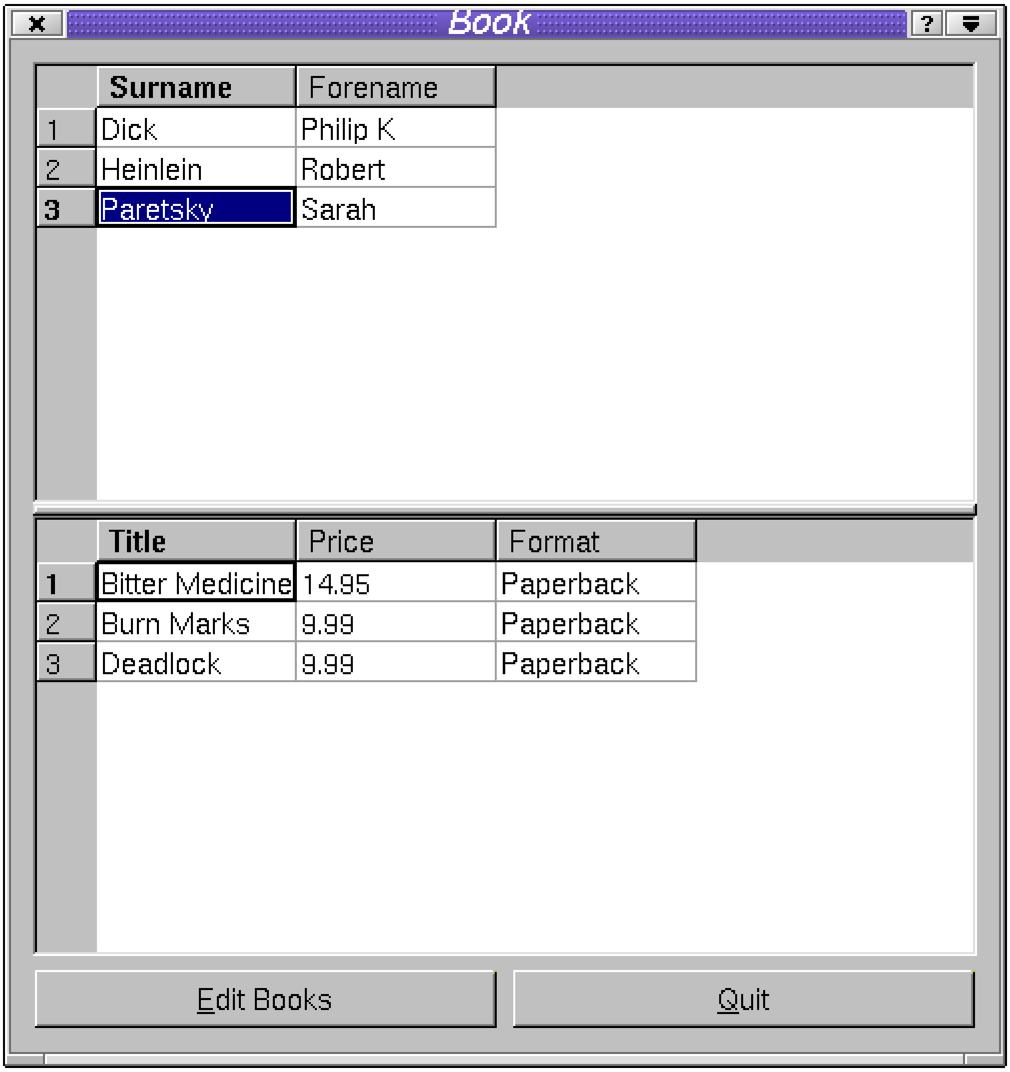


그림 1-66. Book응용프로그람

- 실레스키마

이 절의 모든 실레들은 qt/tools/designer/examples/book/book.sql파일에서 정의되는 표와 보기, 레코드를 사용한다. 이 파일은 PostgreSQL 6과 PostgreSQL 7에 의하여 시험하였다. 이 파일에서 SQL을 수정하여 차기의 체계에서 실레자료기지를 다시 창조할수 있다.

스키마CREATE TABLE문

```
CREATE TABLE author ( id integer primary key, forename varchar(40),
    surname varchar(40) );
CREATE TABLE book ( id integer primary key, title varchar(40),
    price numeric(10,2), authorid integer, notes varchar(255) );
CREATE TABLE sequence ( tablename varchar(10), sequence numeric);
```

book표는 실레의 목적에 맞게 간소화된다. 도서를 단일한 저자(authorid)와 련결할수 있으며 ISBN마당이 없다. Sequence표는 실레표들에 대한 유일색인값을 생성하는데 사용된다. SQL자료기지들은 흔히 더 최적인 풀이로 되는데 아주 류사한 렬(실레로 CREATE SEQUENCE지령의 사용)을 창조하기 위한 자기의 방법을 제공한다. 이식을 위하여 실레들은 대부분의 SQL자료기지들과 작업하는 sequence표를 사용한다.

1. 자료기지련결의 설정

우리가 고려해야 하는 자료기지련결의 두가지 견해가 있다. 첫째로, Qt Designer자체에서 사용하려고 하는 련결이다. 둘째로, 우리가 창조하는 응용프로그람에서 사용하려고 하는 련결이다.

1) Qt Designer의 련결설정

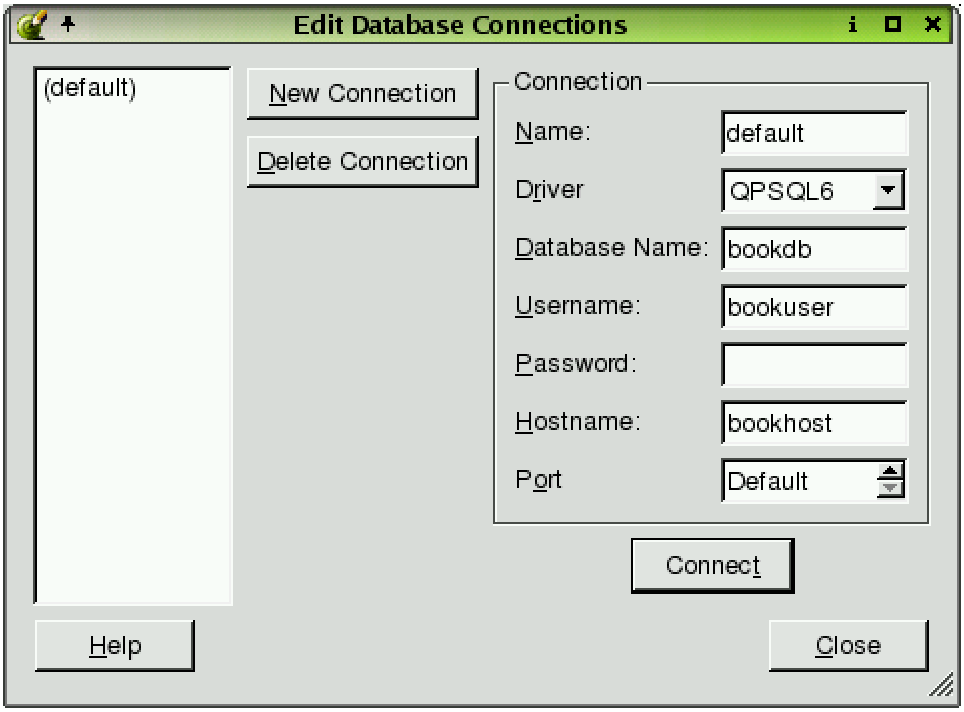


그림 1-67. Edit자료기지련결대화칸

차림표띠에서 Project|Database Connections를 선택한다. Edit Database Connections대화칸이 나타난다. New Connection을 선택한다. 단일자료기지를 사용하는 응용프로그람에 대하여 기정련결이름(default)을 사용하는것이 가장 편리하다. 1개이상의 자료기지를 사용한다면 매개가 유일한 이름으로 주어져야 한다. 구동프로그람은 Driver복합칸으로부터 선택되여야 한다. 자료기지이름은 Database Name복합칸에서 사용하거나 입력해야 한다. 자료기지이름, 사용자이름, 암호, 주콤퓨터이름과 포구는 자료기지체계관리자에 의하여 제공되여야 한다. Connection정보를 완성되였을 때 Connect를 선택한다. 련결이 이루어지면 련결이름이 대화칸의 왼쪽 목록칸에 나타난다. 이제는

대화칸을 닫을수 있으며 련결설정은 그것들을 변경하거나 삭제하거나 Qt Designer로부터 완료할 때까지 효력을 발휘한다.

경고: 현존하는 SQLite자료기지를 사용한다면 Database Name마당에 지정하는 이름이 현존자료기지파일과 같지 않아야 한다는것을 담보해야 한다. Qt Designer는 자료기지용으로 주어지는 이름을 리용하여 환경파일을 창조하고 같은 이름의 현존파일을 다시 써넣는다.

Qt Designer는 qmake프로젝트파일에 자료기지련결설정을 기억할수 있다. 새 프로젝트를 창조한다. 실례로 File|New를 선택하고 C++ Project그림기호를 선택하여 Project Settings대화칸을 연다. 생략단추를 찰칵하여 Save As대화칸을 열고 프로젝트의 등록부로 이행한다. (필요하다면 생성한다.) 프로젝트의 등록부안에 있는가 확인하고 프로젝트이름을 book.pro라고 입력한다. Save단추를 찰칵하여 Project Settings대화칸로 돌아와서 OK를 찰칵한다. 다음번에 Qt Designer가 기동할 때 개별적인 .ui파일들을 열지 않고 .pro프로젝트파일을 열면 Qt Designer는 자동적으로 프로젝트의 련결설정을 재적재한다. 련결을 능동으로 하기 위하여 Project|Database Connections을 선택한다. 프로젝트파일에 이미 보관된 련결들은 왼쪽 목록칸에 표시된다. 사용하려는 련결을 선택하고 Connect를 선택한다. 이제는 이 련결을 사용하며 실례로 QDataTable들을 미리보기하는데 사용한다. 또한 프로젝트파일의 열기는 Qt Designer가 프로젝트와 련결된 폼들의 목록을 Project Overview창문으로 적재하게 한다. 앞으로의 설명에서는 프로젝트파일들을 사용하며 항상 Qt Designer에서 작업할 때 사용할수 있는 련결이 있도록 하기 위하여 Connect를 선택한것으로 가정한다.

2) 응용프로그람을 위한 련결의 설정

창조한 응용프로그람들은 SQL자료기지에 대한 자체의 련결을 만들어야 한다. 자기코드에서 기초로 사용할수 있는 실례함수 createConnections()를 제공한다.

```
bool createConnections()
{
  // create the default database connection
  QSqlDatabase *defaultDB = QSqlDatabase::addDatabase( "QPSQL7" );
  if ( ! defaultDB ) {
    qWarning( "Failed to connect to driver" );
    return FALSE;
  }
  defaultDB->setDatabaseName( "book" );
  defaultDB->setUserName( "bookuser" );
  defaultDB->setPassword( "bookpw" );
```

```
    defaultDB->setHostName( "bookhost" );
    if ( ! defaultDB->open() ) {
        qWarning( "Failed to open books database: " +
                  defaultDB->lastError().driverText() );
        qWarning( defaultDB->lastError().databaseText() );
        return FALSE;
    }

    return TRUE;
}
```

사용하려는 구동프로그람의 이름을 넘기여 addDatabase()를 호출한다. 그다음 set함수들을 호출하여 련결정보를 설정한다. 끝으로 련결을 열려고 시도한다. 성공하면 TRUE를 돌려주며 그렇지 않으면 오유정보를 출력하고 FALSE을 돌려준다.
(qt/tools/designer/examples/book/book1/main.cpp에서 발취.)

```
int main( int argc, char *argv[] )
{
    QApplication app( argc, argv );

    if ( ! createConnections() )
        return 1;

    BookForm bookForm;
    app.setMainWidget( &bookForm );
    bookForm.show();

    return app.exec();
}
```

이 절에서 제시한 모든 실례들은 main.cpp파일에서 QApplication객체를 창조한 다음 createConnections()를 호출하여 기정련결을 사용할수 있게 만든다. 여러개의 자료기지에 련결하려고 한다면 addDatabase()의 2인수형식을 사용하며 거기에 구동프로그람의 이름과 유일식별자를 넘긴다.

자료기지련결에 대한 참고를 유지할 필요는 없다. 단일한 자료기지련결을 사용한다면 이것은 기정련결으로 되고 자료기지함수들은 이 련결을 자동적으로 사용한다. 우리는 항상 QSqlDatabase::database()를 호출하여 임의의 련결에로의 지적자를 얻을수 있다.

Qt Designer를 사용하여 main.cpp파일을 창조한다면 이 파일은 createConnections()를 포함하지 않는다. 그것이 자료기지련결에 사용자이름과 암호를 요구하기때문에 이 기능을 포함하지 않으며 이것들을 우의 단순한 실례함수와 다르게 취급하는것을 좋아할수 있다. 결국 Qt Designer에서 정확히 미리보기되는 응용프로그람은 자기의 자료기지련결함수를 실현하지 않는한 실행되지 않는다.

2. QDataTable의 리용

QDataTable을 임의의 폼에 배치하여 자료기지표와 보기의 열람을 제공할수 있다. QDataTable은 in-place에서 가령 세포자체안에서 레코드들을 갱신하거나 삭제하는데 사용될수 있다. QDataTable을 경유한 레코드삽입은 보통 실례를 위한 주열쇠를 생성하거나 기정값을 제공할수 있도록 primeInsert()신호에로의 련결을 요구한다. 폼보기를 사용하여(여러개의 표과 보기에서 자료를 결합하여) 레코드를 제시하려면 여러개의 QDataBrowser와 QDataView를 사용할수도 있다.

1) 자료기지표의 고속보기

다음의 실례는 이 절의 다른 모든 실례들처럼 프로젝트이름 book를 가지며 book.sql스크립트에 의해 만들어지는 자료기지를 사용한다. 이 절에 따라 book응용프로그람을 한걸음씩 건설한다. 초기에 보여준 qt/tools/designer/examples/book/book1/main.cpp파일을 창조하거나 복사한다. 첫번째 실례를 위한 프로젝트파일은 qt/tools/designer/examples/book/book1/book.pro이다. 새로운 프로젝트를 File|New를 선택하는것으로 시작하고 C++ Project그림기호를 선택하여 Project Settings대화칸을 펼친다. 생략단추를 찰칵하여 Save As대화칸을 열고 프로젝트의 등록부(필요하다면 만든다.)로 이행한다. 프로젝트의 등록부에 있는것을 확인하고 프로젝트이름을 book.pro라고 입력한다. Save단추를 찰칵하여 Project Settings대화칸로 돌아와서 OK를 찰칵한다. 이제는 Project|Database Connections를 선택한다. 자기의 자료기지에 적당한 련결정보를 기입하고 Connect를 찰칵한다. 련결이름이 왼쪽 목록칸에 나타나야 한다.대화칸을 닫는다.

그러면 우리 자료기지표중의 하나에 련결되는 QDataTable을 가진 새로운 폼을 창조한다.

File|New를 선택한다. New File대화칸은 선택가능한 많은 폼형판을 제시한다. Dialog폼을 선택하고 OK를 찰칵한다. 지금 File|Save를 선택한다. 파일이름을 book.ui라고 부른다.

- QDataTable의 설정

폼에 QDataTable창문부품을 배치하기 위하여 Tools|Views|DataTable을 선택하거나 DataTable 도구띠단추를 찰칵한다. 폼을 선택하면 Data Table Wizard가 나타난다.

① Database Connection and Table위자드페지는 만일 존재하지 않으면 련결을 설정하고 QDataTable을 위한 표 또는 보기를 선택하는데 사용된다.

왼쪽 Database Connection 목록칸에서 사용하려는 련결 례를 들면 (default)를 선택한다. 사용가능한 표와 보기는 오른쪽 Table목록칸에 나타난다. author표를 선택하고 Next단추를 찰칵한다.

② Displayed Fields위자드페지는 QDataTable에서 표시할 마당들과 그 순서를 선택하는 수단을 제공한다. 기정으로 주열쇠(만일 있으면)이외의 모든 마당은 Displayed Fields목록칸에 있다. 왼쪽과 오른쪽의 푸른 화살단추는 Displayed Fields와 Available Fields목록칸사이에서 마당을 이동하는데 사용될수 있다. 푸른색의 올리와 내리화살단추는 표시된 마당의 표시순서를 선택하는데 사용된다.

기정설정을 요구하므로 단지 Next단추를 찰칵한다.

③ Table Properties위자드페지는 QDataTable의 자료기지관련속성을 호출하는데 편리를 제공한다.

Confirm Deletes검사칸이 선택되였는가 확인하고 Next단추를 찰칵한다.

④ SQL위자드페지는 QDataTable의 Filter와 Sort속성들을 설정하는데 사용된다. Filter는 SQL WHERE절이다(단어 WHERE없이). 례를 들면 단지 성이 P로 시작되는 저자를 표시하기 위하여서는 title LIKE 'P%'를 입력한다. 려파기를 비워놓는다. Available Fields목록칸은 모든 마당을 표시한다. Sort By목록칸은 QDataTable을 분류하려는 마당들과 분류방법(ASCending 혹은 DESCending)을 표시한다. 좌우의 푸른 화살은 2개의 목록칸사이에서 마당을 이동하는데 사용된다. 우아래의 푸른 화살은 Sort By목록칸안에서 마당을 상하로 움직인다. ASC 또는 DESC설정은 sort order도구띠단추로 교체된다.

Sort By목록칸으로 성과 이름마당을 이동하고 Next를 찰칵한다.

⑤ Finish위자드페지는 되돌아와서 임의의 설정을 변경할수 있는 기회를 준다. QDataTable의 속성을 통하여 그것들을 후에 변경할수 있으므로 위자드를 완료할수 있다.

Finish를 선택한다.

표는 기정렬이름으로 표식된 렬들을 가지는 폼에 나타난다. 설정을 변경하려면 그 대부분은 속성창문에서 사용할수 있다. 현시이름, 그것들이 기초하는 마당과 렬들의 표시순서는 QDataTable을 오른쪽 단추로 누르고 Edit를 왼쪽 단추로 눌러서 표시되는 Edit Table대화칸(후에 설명)를 사용하여 변경할수 있다.

- 폼의 배치

폼을 선택하고 Lay Out Vertically도구띠단추를 찰칵한다. 지금 Preview|Preview Form을 선택하여 폼을 실행하면 표는 모든 레코드를 자동적으로 표시한다.

실행가능한 응용프로그람에서 생성한 폼으로 절환하기 위하여서는 main.cpp파일을 프로젝트파일에 추가하고 프로젝트를 구축해야 한다. 또한 일부 이름을 변경하여 리해하

기 쉽게 해야 한다.

① 폼을 선택하고 이름을 BookForm로, 제목을 Book로 변경한다. QDataTable을 선택하고 이름을 AuthorDataTable로 변경한다.

② File|Save All을 선택한다.

③ 평본문편집기에서 프로젝트파일 실례로 book.pro를 열고 파일의 끝에 다음의 행 SOURCES += main.cpp을 추가한다.

④ qmake를 실행하여 make파일을 생성한다. 실례로 qmake -o Makefile book.pro. 그다음 book프로그람을 구축하고 실행한다.

이 실례는 QDataTable을 사용하여 자료기지의 표나 보기의 내용을 보여주는것이 얼마나 쉬운가를 보여준다. 방금 건설한 응용프로그람을 사용하여 저자레코드를 갱신하고 삭제할수 있다. 그 다음의 실례에서 삽입과 주-세부관계설정, 구멍내기와 외부열쇠검색을 설명한다.

- 외부열쇠에 대한 알아두기

대부분의 관계형자료기지에서 표는 다른 표에로의 외부열쇠들로 되여있는 마당을 포함한다. book자료기지실례에서 book표의 authorid는 저자표에로의 외부열쇠이다. 말단사용자에게 폼을 제시할 때 보통 외부열쇠 그 자체가 아니라 그와 련결된 본문을 보려고 한다. 마찬가지로 도서정보를 보여줄 때 저자ID가 아니라 저자의 이름을 표시하려고 한다. 많은 자료기지들에서 이것은 보기에 의해 이루어질수 있다.

2) QDataTable에 레코드의 삽입

관계형자료기지에 대한 레코드삽입은 보통 표에서 레코드를 유일하게 식별하는 주열쇠값의 생성을 요구한다. 또한 자주 어떤 마당용의 기정값을 창조하여 사용자의 일감을 줄이려고 한다. QDataTable의 primeInsert()신호를 받아들이는 처리부를 만들고 유일한 주열쇠를 가지는 QSqlRecord삽입완충기를 구성한다.

① Edit|Slots를 선택하여 Edit Functions대화칸을 연다. New Function을 선택한 다음 Function Properties의 Function행편집칸에 처리부이름이름 primeInsertAuthor(QSqlRecord*)를 입력한다. OK를 찰칵한다.

② Connect Signals/Slots도구띠단추를 찰칵하고 AuthorDataTable을 선택하고 폼으로 끌고가서 마우스를 놓는다. 그러면 Edit Connections대화칸이 나타난다. primeInsert()신호와 primeInsertAuthor()처리부를 각각 선택하고 련결을 만든다. OK를 찰칵한다.

③ Object Explorer창문의 Members타브를 선택한다. (필요하다면 Window|Views|Object Explorer를 선택하여 창문을 표시한다.) primeInsertAuthor()처리부를 선택하면 편집기창문이 나타난다.

④ BookForm::primeInsertAuthor()처리부를 변경하여 파라메터이름을 지정하고

필요한 작용을 처리한다.

```
void BookForm::primeInsertAuthor( QSqlRecord * buffer )
{
    QSqlQuery query;
    query.exec( "UPDATE sequence SET sequence = sequence + 1 "
                        "WHERE tablename='author';" );
    query.exec("SELECT sequence FROM sequence WHERE tablename='author';");
    if ( query.next() ) {
        buffer->setValue( "id", query.value( 0 ) );
    }
}
```

QSqlQuery객체는 저자표에 대한 유일한 sequence번호를 증가시키고 얻는데 사용된다. 신호는 우리에게 삽입완충기에의 지적자를 넘기였고 그다음 우리가 얻은 값(즉 다음 순서렬번호)을 완충기의 ID마당에 넣었다. (흔히 SQL자료기지가 자연적인 순서렬 함수를 지원한다. 여기에서 사용되는 방법은 생산체계에 잘 어울리지 않고 실례의 목적을 위해서뿐이다. 코드에서 유일한 열쇠를 생성하는 방법에 대한 자세한 내용은 자기 자료기지의 참고문서를 참고하시오. 많은 경우에 자료기지는 그것들을 자동적으로 생성할수 있다. 또는 자료기지는 순서렬을 다루기 위하여 특별한 구문을 제공할수도 있다.)

응용프로그람을 다시 건설하면 UPDATE와 DELETE뿐아니라 INSERT를 지원한다. 간단히 기정값(실례로 필요하다면 날자마당에 오늘의 날자)을 삽입하기 위한 보충적인 코드를 추가할수 있다.

열람은 레코드를 선택하고 화살표건을 사용하여 진행된다. 레코드가 강조표시되면 그것을 편집할수 있다. Insert건을 눌러서 새로운 레코드를 삽입하고 F2을 눌러서 현재 레코드를 갱신하며 Del건을 눌러 현재의 레코드를 삭제한다. 이러한 모든 조작은 즉시 일어난다. 사용자들은 QDataTable의 confirmEdits속성을 True로 설정하여 자기들의 편집을 확인할 기회를 얻을수 있다. confirmEdits속성이 True라면 모든 삽입과 갱신, 삭제에 대해 사용자확인이 요구된다. 더 상쾌한 조종을 위하여 confirmInsert와 confirmUpdate, confirmDelete속성을 개별적으로 설정할수 있다.

- QDataTable사용자대면부교제

QDataTable들을 위한 기정의 사용자대면부조작은 다음과 같다.

• 사용자들은 마우스로 흐름띠를 눌러 레코드들을 선택하고 옮길수 있다. 또한 건반의 이동건 실례로 왼쪽화살표, 오른쪽화살표, 올리화살표, 내리화살표, PageUp, PageDown, Home, End를 사용할수 있다.

• INSERT는 레코드를 오른쪽 단추로 선택하고 Insert를 누르거나 또는 Ins(삽입)

건을 누르는것으로 시작된다. 사용자는 Tab와 Shift+Tab를 사용하여 마당들사이에서 이동한다. 사용자가 마지막 마당으로부터 떨어져서 Enter나 Tab를 누르면 INSERT가 일어난다. autoEdit가 TRUE이고 사용자가 다른 레코드로 항행하면 삽입이 일어난다. INSERT는 Esc(Escape)를 누르면 취소된다. autoEdit가 FALSE라면 다른 레코드에로의 항행은 INSERT를 취소한다. confirmInsert를 TRUE로 설정하면 INSERT할 때마다 사용자에게 확인할것을 요구한다.

• UPDATE는 레코드를 오른쪽 단추로 선택하고 Update를 누르거나 F2를 누르는것으로 시작된다. 사용자가 마지막 마당으로부터 떨어져서 Enter나 Tab를 누르면 갱신이 일어난다. autoEdit가 TRUE이고 사용자가 다른 레코드로 항행하면 갱신이 일어난다. UPDATE는 Esc를 눌러서 취소된다. autoEdit가 FALSE일 때 다른 레코드에로 항행하면 UPDATE를 취소한다. confirmUpdate를 TRUE로 설정하면 UPDATE할 때마다 사용자에게 확인할것을 요구한다.

• DELETE는 레코드를 오른쪽 단추로 찰칵하고 Delete를 누르거나 Del(삭제)건을 누름으로써 이루어진다. confirmDelete를 TRUE로 설정하면 DELETE할 때마다 사용자에게 확인할것을 요구한다.

필요하다면 기정동작을 프로그람적으로 변경할수 있다.

3) 두개 표의 련결

흔히 자료기지는 관련된 표들의 쌍을 가지고있다. 례를 들면 청구서표는 청구서항목표가 보관하는 실제 청구서항목이 아니라 청구서의 번호와 날자, 손님들을 목록할수 있다. book응용프로그람에서는 저자표를 통하여 열람하는데 사용할수 있는 첫째 QDataTable과 저자들이 저술한 도서를 보여주기 위한 두번째 QdataTable을 가진다.

Qt Designer가 이미 열려있지 않으면 book프로젝트를 연다. 이 프로젝트를 수정하여 저자표를 도서표와 련결하는 2개의 QDataTable을 보여준다.

① 저자 QDataTable을 선택하고 Break Layout도구단추를 찰칵한다.

② 단지 폼의 우의 절반을 차지하도록 DataTable의 크기를 변경한다.

③ DataTable도구단추를 찰칵하고 폼의 아래절반을 선택한다. Table Wizard가 나타난다. (이 위자드는 자료기지표의 고속보기에서 설명된다.) 다음과 같이 조작한다.

• 사용하고있는 련결을 선택하고 도서표를 선택한다. Next단추를 찰칵한다.

• 그것들을 보이게 하려고 하지 않으므로 authorid를 확인하고 id마당은 화살단추를 사용하여 Available Fields목록칸으로 움직이게 된다. 제목마당을 Displayed Fields의 꼭대기로 이동하고 notes마당우의 price마당을 이동한다. Next단추를 찰칵한다.

• Table Properties페지에서 Read Only검사칸을 선택하고 Next단추를 찰칵한다.

• SQL페지에서 Filter(WHERE절)를 비워놓는다. 제목마당을 Sort By목록칸으로 이동하고 Next를 찰칵한다. Finish를 찰칵한다.

• QDataTable의 이름을 BookDataTable로 변경한다.

④ Shift를 누른 상태에서 최상우의 QDataTable을 선택하고 기타 QDataTable들을 선택한 다음 Lay Out Vertically(in Splitter)도구띠단추를 찰칵한다.

⑤ 폼을 선택하고 Lay Out Vertically도구띠단추를 찰칵한다.

Preview|Preview Form을 선택하여 폼을 미리보기한다. 모든 저자는 최상위의 QDataTable에 표시되고 모든 도서는 아래의 QDataTable에 표시된다. 그러나 바닥의 QDataTable에서 표시되는 현재 선택된 저자의 도서들만 요구한다. 저자표에서 선택된 저자에 따라서 도서표에서 레코드들을 려과함으로써 이것을 취급한다.

- 표편집기의 사용

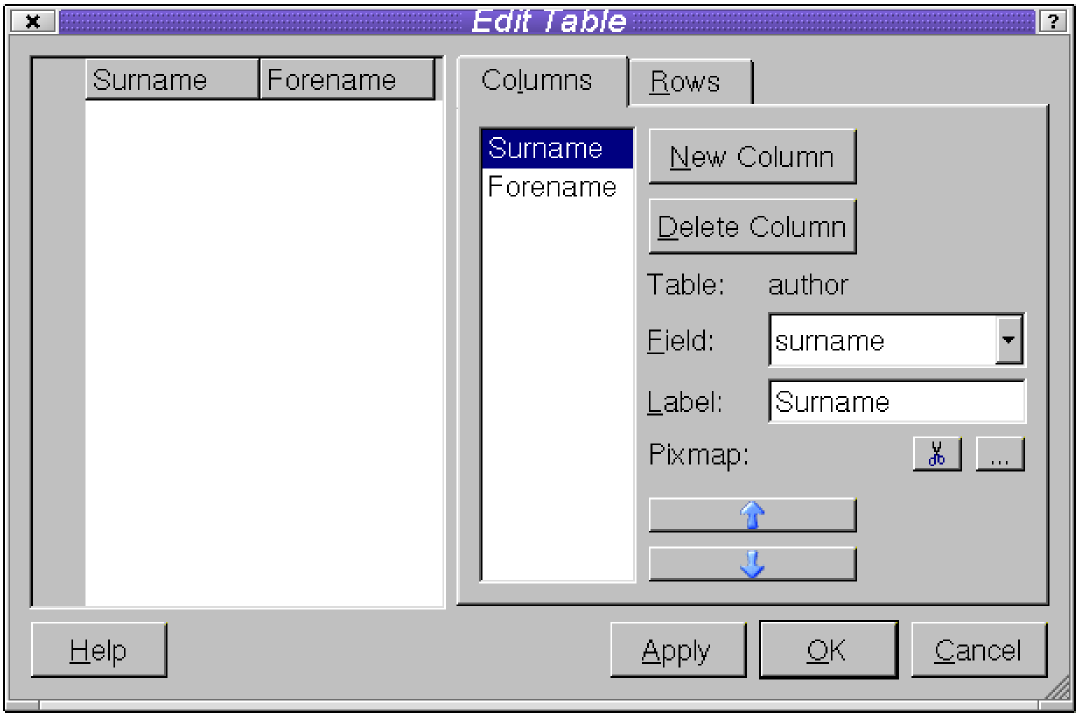


그림 1-68. Edit Table대화칸

SQL Table위자드를 사용하여 QDataTable을 창조하고 설정한다. 다른 Qt Designer창문부품처럼 그 속성을 Properties창문에서 변경할수도 있다. 속성에 기초한 일부 렬과 행은 Edit Table대화칸을 사용하여 변경할수 있다. 이 대화칸은 QDataTable을 오른쪽 단추로 선택하고 Edit차림표항목을 왼쪽 단추로 눌러서 펼친다. Edit Table대화칸의 오른쪽 절반은 표시하려는 마당들과 그 순서, 표식들을 선택하는 곳이다. 렬을 만드는 절차는 다음과 같다.

① New Column단추를 찰칵한다.

② Field복합칸을 펼치여 사용가능한 마당을 표시한다.

③ 포함하려는 마당을 선택한다.

④ 기정값이 적당하지 않으면 선택적으로 Label을 편집한다.

⑤ Pixmap생략(…)단추를 찰칵하고 렬표식자의 왼쪽에 표시하려는 픽스매프를 선택한다. (생략단추는 pixmap 또는 iconSet속성에 의해 Properties목록의 Value부분을 선택할 때 나타난다.)

추가하려는 매개 렬에 대하여 우의 단계를 반복한다. 모든 마당을 추가하였다면 푸른색의 올리와 내리 화살단추를 사용하여 그 순서를 바꿀수 있다. 임의의 점에서 Apply를 눌러서 표의 표시를 확인할수 있다. 끝으로 OK단추를 찰칵하여 설정한 속성을 보관한다. 항상 표편집기로 되돌아와서 이 설정을 변경할수 있다.

- 한개 QDataTable을 다른것에 의하여 려파하기

도서표의 레코드를 려파하기 위하여 저자 QDataTable의 currentChanged()신호를 받아들이고 BookDataTable의 려파기를 변경한다.

① Edit|Slots를 선택한다. Edit Functions대화칸에서 New Function을 선택하고 newCurrentAuthor(QSqlRecord*)라는 처리부이름을 입력한다. OK를 찰칵한다.

② Edit|Connections을 선택하여 View and Edit Connections대화칸을 연다. AuthorDataTable의 currentChanged()신호를 폼의 newCurrentAuthor()처리부에 련결하는 새로운 련결을 만든다. OK를 찰칵한다.

③ Object Explorer창문의 Members타브를 선택한다. (필요하다면 Window|Views|Object Explorer를 선택하여 창문을 표시한다.) newCurrentAuthor()처리부를 선택하면 편집기창문이 나타난다.

④ BookForm::newCurrentAuthor()처리부를 변경하여 파라메터이름을 지정하고 필요한 작용을 처리한다.

```
void BookForm::newCurrentAuthor( QSqlRecord *author )
{
    BookDataTable->setFilter( "authorid=" + author->value( "id" ).toString() );
    BookDataTable->refresh();
}
```

지금 필요한것은 BookDataTable의 려파기를 바꾸고 려파기의 결과를 보여주도록 QDataTable을 재생하는것이다.

- Drilldown용 대면부의 준비

이제는 저자들을 열람하고 편집할수 있으며 BookDataTable에서 그들의 책을 볼수 있다. 다음 절에서는 QDataBrowser를 탐구한다. 이것은 책을 편집할수 있는 대화칸을 펼칠수 있게 한다. 현재는 도서편집대화칸을 여는데 사용하는 기본BookForm에 단추들을 추가한다.

① 폼을 선택하고 Break Layout도구띠단추를 찰칵한다. 바닥에서 단추들에 필요한

자리를 폼에 만들어주기 위하여 크기를 변경한다.

② 2개 단추를 폼의 바닥에 추가한다. 이름과 표식자를 다음과 같이 변경한다.

EditPushButton -- &Edit Books

QuitPushButton -- &Quit

Shift건을 누르면서 두개의 단추를 누르고(즉 Shift를 누르면서 단추들을 찰칵한다.) Lay Out Horizontally도구띠단추를 찰칵한다. 폼을 선택하고 Lay Out Vertically도구띠단추를 찰칵한다.

이제는 Quit단추에 기능을 제공한다. Edit|Connections을 선택한 다음 Quit단추의 clicked()를 폼의 accept()처리부에 련결하다. OK를 찰칵한다.

3. QDataBrowser와 QDataView의 리용

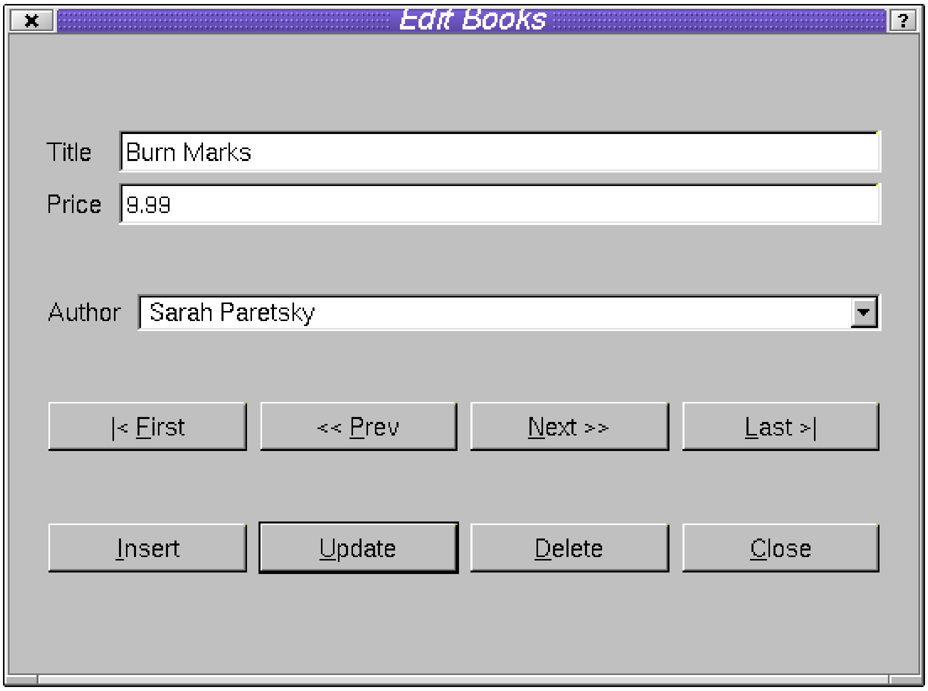


그림 1-69. Book응용프로그람의 Edit Books대화칸

- QDataBrowser설정

이제는 새로운 폼을 창조하여 사용자에게 책레코드를 편집하게 한다. New도구띠단추를 찰칵하고 New File대화칸에서 Dialog형판을 선택하고 OK를 찰칵한다. 폼이름을 EditBookForm으로, 제목을 Edit Books로 변경한다. Save도구띠단추를 찰칵하고 파일을 editbook.ui라고 부른다. 이제는 QDataBrowser를 추가하여 책레코드들을 보여줄수 있는 폼이 있다.

① Data Browser도구띠단추를 찰칵하고 폼을 선택한다. Data Browser Wizard가 나타난다.

② Database Connection and Table위자드페지는 존재하지 않는 련결을 설정하고 QDataBrowser를 위한 표 또는 보기를 선택하는데 사용된다.

Database Connection목록칸에서 사용하려는 련결 례를 들면 "(default)"을 선택한다. 사용가능한 표와 보기들이 Table목록칸에 나타난다. 책표를 선택하고 Next단추를 찰칵한다.

③ Displayed Fields위자드페지는 QDataBrowser에서 표시하려는 마당들과 순서를 선택하는 수단을 제공한다. 기정으로 주열쇠(만일 있으면)이외의 모든 마당은 오른쪽 Displayed Fields목록칸에 있다. 좌우의 푸른 화살단추들은 Displayed Fields와 Available Fields목록칸사이에서 마당을 움직이는데 사용될수 있다. 올리와 내리의 푸른색화살단추는 표시된 마당들의 현시순서를 선택하는데 사용된다.

우리는 폼에서 외부열쇠마당을 확인하려고 하지 않으므로 그것을 Available Fields목록칸으로 이동한다. 또한 제목마당을 Displayed Fields목록의 꼭대기로 이동한다. Next단추를 찰칵한다.

④ Navigation and Editing위자드페지는 폼에 나타나야 하는 항행 및 단추편집을 선택하게 한다.

기정값을 받아들이고 Next단추를 찰칵한다.

⑤ SQL위자드페지는 QDataBrowser의 Filter와 Sort속성들을 설정하는데 사용된다. Filter는 SQL **WHERE** 절이다. 례를 들면 단지 50원미만인 책들만 표시하기 위해서는 WHERE없이 **price < 50**라고 입력한다. 려파기를 비워놓는다. Available Fields목록칸은 모든 마당을 목록한다. Sort By목록칸은 QDataBrowser가 분류하는 마당들과 분류방향(ASCending 혹은 DESCending)을 표시한다. 좌우의 푸른 화살표들은 2개의 목록칸사이에서 마당들을 움직이는데 사용된다. 푸른색의 올리와 내리화살표들은 Sort By목록칸안에서 마당을 상하로 이동한다. ASC 또는 DESC설정은 분류순서단추로 변경된다.

Sort By목록칸으로 제목마당을 옮기고 Next를 찰칵한다.

⑥ Layout위자드페지는 폼의 처음배치를 지정하는데 사용된다.

Number of Columns를 1로 변경하고 Next단추를 찰칵하고 Finish를 찰칵한다.

⑦ QDataBrowser는 폼우에 나타난다. 폼의 크기를 변경하여 더 작게 한다. QDataBrowser를 선택하고 Break Layout도구띠단추를 찰칵한다. 단추들을 누르고 Break Layout도구띠단추를 찰칵한다. 본문 &Close를 가지는 PushButtonClose라고 부르는 다른 단추를 추가하고 그것을 Delete단추의 오른쪽에 배치한다.

⑧ Shift를 누르면서 Insert와 Update, Delete, Close단추를 찰칵한 다음 Lay Out Horizontally도구띠단추를 찰칵한다. QDataBrowser를 선택하고 도구띠단추에서 Lay Out in a Grid를 선택한다. 끝으로 폼을 선택하고 Lay Out Vertically도구띠단

추를 찰칵한다. 이제는 QDataBrowser를 선택하고 이름을 BookDataBrowser로 변경한다.

⑨ Qt Designer는 필요한 코드(적당한 유표, 분류와 려파기코드)를 생성하여 열람프로그람을 사용할수 있게 한다.

폼에 대한 더 상쾌한 조종을 위하여서는 자기의 자료기지유표를 창조한다. 그리하여 Properties창문에서 BookDataBrowser의 frameworkCode속성을 FALSE로 설정하고 유표를 위한 여분의 코드를 Qt Designer가 생성하는것을 방지한다.

- QDataBrowser사용자대면부교제

QDataBrowser를 위한 사용자대면부동작은 처리부와 신호들을 련결하여 창조한다. 제공되는 처리부들은 다음과 같다.

- 편집용의 insert()와 update(), del()
- 항행용의 first()와 next(), prev(), last()
- 자료기지로부터 유표를 재생하기 위한 refresh()
- 유표의 편집완충기로부터 자료를 읽기 위한 readFields()과 유표의 편집완충기에 폼의 자료를 쓰기 위한 writeFields()
- 폼의 값들을 지우기 위한 clearValues()

Qt Designer의 QDataBrowser위자드를 사용한다면 항행 및 편집을 위한 기정단추모임을 창조하는 선택이 주어진다. 이 단추들의 동작은 다음의 기능을 제공하며 우에서 서술한 처리부에 의하여 설정된다.

- INSERT는 Ins(Insert)건을 누르는것으로 시작된다. 사용자는 Tab와 Shift+Tab를 사용하여 마당들사이에서 움직인다. 사용자가 Update단추를 누르면 INSERT가 발생하며 사용자는 방금 삽입한 레코드로 간다. 사용자가 Insert단추를 두번째로 누르면 INSERT가 발생하고 새로운 삽입이 시작된다. autoEdit가 TRUE이고 사용자가 다른 레코드로 항행하면 INSERT가 발생한다. INSERT는 Esc건을 누르거나 Del(삭제)건을 누름으로써 취소된다. autoEdit가 FALSE일 때 다른 레코드로 항행하면 INSERT를 취소한다. confirmInsert를 TRUE로 설정하면 사용자에게 각 INSERT를 확인할것을 요구한다.
- 사용자가 레코드를 항행할 때마다 UPDATE는 자동적으로 시작된다. 사용자가 Update단추를 누르면 갱신이 일어난다. autoEdit가 TRUE이고 사용자가 다른 레코드로 항행하면 갱신이 일어난다. UPDATE는 Esc건을 누르거나 Del단추를 누름으로써 취소된다. autoEdit가 FALSE일 때 다른 레코드에로의 항행은 UPDATE를 취소한다. confirmUpdate를 TRUE로 설정하면 사용자에게 각 UPDATE를 확인할것을 요구한다.
- DELETE는 Del건을 누르는것으로 이루어진다. confirmDelete를 TRUE로 설정하면 사용자에게 각 DELETE를 확인할것을 요구한다.

- 구멍내기처리

이제는 책레코드를 편집하기 위한 폼이 있다. 사용자가 Edit Books단추를 찰칵할 때 폼를 기동하여 BookDataTable에서 선택한 레코드를 항행해야 한다. 또한 외부열쇠(실례로 authorid)를 편집하는 수단을 제공해야 한다.

① Edit Books단추의 clicked()신호를 련결할 새로운 처리부를 만들어야 한다. Book폼을 선택하여 Qt Designer의 능동폼으로 한다. Edit Functions대화칸을 호출하고 editClicked()라는 이름의 새로운 기능을 창조한다. 이제 Edit|Connections를 선택한다. Edit Books단추의 clicked()신호를 폼의 editClicked()처리부에 련결한다. OK를 찰칵하여 대화칸을 닫는다.

② Object Explorer창문에서 Members를 선택하고 editClicked함수를 선택한다. 그것을 다음과 같이 변경한다.

```
void BookForm::editClicked()
{
    EditBookForm *dialog = new EditBookForm(this, "Edit Book Form", TRUE);
    QSqlCursor cur( "book" );
    dialog->BookDataBrowser->setSqlCursor( &cur );
    dialog->BookDataBrowser->setFilter( BookDataTable->filter() );
    dialog->BookDataBrowser->setSort(QSqlIndex::fromStringList(
        BookDataTable->sort(), &cur ) );
    dialog->BookDataBrowser->refresh();
    int i = BookDataTable->currentRow();
    if ( i == -1 ) i = 0; // Always use the first row
    dialog->BookDataBrowser->seek( i );
    dialog->exec();
    delete dialog;
    BookDataTable->refresh();
}
```

이전과 같이 대화칸을 만든다. 또한 책표우에 유표를 창조하고 대화칸의 QDataBrowser, BookDataBrowser를 설정하여 새 유표를 사용한다. 기본폼의 책 QDataTable에 적용되는 것들로 QDataBrowser의 려과기와 분류를 설정한다. QDataBrowser를 재생하고 사용자가 기본폼에서 보는것과 같은 레코드를 탐색한다. 그 다음 대화칸을 실행하고 사용자가 완료했을 때 그것을 삭제한다. 끝으로 기본폼에서 BookDataTable을 갱신하여 대화칸에서 만들어진 변경을 반영한다.

③ 우리 코드가 editbook.h에서 선언된 클라스와 QDataBrowser를 참고하므로 2개의 머리부파일을 추가한다. BookForm을 선택하고 Object Explorer창문의 Members타브를

선택한다. Includes (In Declaration)항목을 오른쪽 단추로 선택하고 New를 찰칵하고 editbook.h라고 입력한다. 이제 둘째 머리부파일 <qdatabrowser.h>를 포함한다.

BookForm에서 저자와 책레코드를 통하여 항행할 때 Edit Books단추를 찰칵하여 Edit Books대화칸을 연다. 비록 대화칸이 책표에서의 UPDATE와 DELETE, 항행을 지원하지만 외부열쇠를 편집할수도 없고 삽입할수도 없다. QDataTable에서와 같은 방법으로 삽입을 취급하고 저자와의 외부열쇠관계를 조종한다.

- QDataBrowser에 삽입

유일한 주열쇠를 삽입할수 있도록 Edit Books폼의 primeInsert()신호를 받기 위한 처리부를 창조한다.

① Edit Books폼를 선택하고 primeInsertBook(QSqlRecord*)라는 이름으로 새로운 Slot를 창조한다.

Edit|Slots를 선택하고 New Function단추를 찰칵하고 Function Properties Function편집칸에서 새로운 함수이름을 입력하고 OK를 찰칵한다.

② BookDataBrowser의 primeInsert()신호를 primeInsertBook()처리부에 련결한다.

Connect Signals/Slots도구띠단추를 찰칵한 다음 BookDataBrowser를 선택하고 폼우로 끌고가서 마우스를 놓는다. 그리고 primeInsert()신호와 primeInsertBook처리부를 선택하고 OK를 찰칵한다.

③ Object Explorer창문에서는 Members를 선택하고 primeInsertBook처리부를 선택한다. 다음과 같이 변경한다.

```
void EditBookForm::primeInsertBook( QSqlRecord * buffer )
{
  QSqlQuery query;
  query.exec( "UPDATE sequence SET sequence = sequence + 1 "
                    "WHERE tablename='book';" );
  query.exec( "SELECT sequence FROM sequence WHERE tablename='book';" );
  if ( query.next() ) {
    buffer->setValue( "id", query.value( 0 ) );
  }
}
```

④ 또한 사용자대면부를 좀 아름답게 하려고 한다. Update단추를 찰칵하고 기정속성을 True로 설정한다. Close단추의 clicked()신호를 EditBookForm의 accept()처리부에 련결한다.

- QDataBrowser에서 외부열쇠조종

Qt의 SQL모듈에서 외부열쇠를 취급하는 2가지 수법을 제공한다. 가장 강력하고 유

연한 수법은 창문부품들의 파생클라스를 만드는것과 속성랴도를 사용하여 창문부품들을 자료기지와 련결하는것이다. 이 수법은 Qt SQL모듈문서(특히 StatusPicker실례)에서 서술된다. Qt Designer에서 전적으로 취할수 있는 더 단순한 수법을 여기서 설명한다.

새로운 마당을 EditBookForm에 추가하여 저자들을 제목과 가격과 함께 편집할수 있게 한다. 시각적으로 설계하였다면 코드를 써서 그것이 모두 작업하게 한다.

① 우선 새 창문부품을 추가한다. BookDataBrowser를 선택하고 Break Layout 도구띠단추를 찰칵한다. 폼의 크기를 더 크게 하고 매개 단추모임을 끌어서 제목와 가격 QLineEdit들아래에 빈자리를 만든다. Text Label도구띠단추를 찰칵하고 폼에서 Price표식자아래를 선택한다. Text Label을 선택하고 본문을 Author로 변경한다. ComboBox도구띠단추를 찰칵하고 폼우에서 가격QLineEdit의 아래를 선택한다. Property Window에서 ComboBox의 이름을 ComboBoxAuthor로 바꾸고 sizePolicy hSizeType를 Expanding으로 바꾼다.

② 대화칸을 배치한다. Shift를 누른 상태에서 Author표식자과 ComboBox를 선택한 다음 Lay Out Horizontally도구띠단추를 찰칵한다. BookDataBrowser를 선택하고 Lay Out in a Grid도구띠단추를 찰칵한다.

코드를 추가하여 ComboBox에 저자이름들을 보관하고 현재 책의 저자에게로 스크롤할수 있게 한다. 또한 책레코드를 삽입하거나 갱신할 때 저자id를 책표의 authorid마당에 넣는다는것을 담보해야 한다. 그것을 처리부에 넣고 신호를 처리부에 련결하여 코드가 정확히 실행되도록 담보한다.

① beforeUpdateBook(QSqlRecord *buffer)와 primeUpdateBook(QSqlRecord *buffer)라는 이름으로 2개의 새로운 처리부를 만든다.(Edit|Slots를 선택하고 Edit Functions대화칸에서 New Function을 선택하고 첫번째 새로운 처리부를 입력한다. New Function을 다시 선택하고 두번째 처리부를 입력하고 OK를 찰칵한다.)

② 사용자가 대화칸을 통하여 항행할 때 새로운 레코드로 옮길 때마다 primeUpdate()신호가 발송된다. 여기에 련결하여 ComboBox의 현시를 갱신할수 있다. 자료기지에서 하나의 레코드가 갱신되거나 삽입되기전에 beforeUpdate() 혹은 beforeInsert()신호가 발송된다. beforeUpdateBook()처리부를 두개의 신호에 련결하여 책의 authorid마당이 정확히 형성되는것을 담보할수 있다.

BookDataBrowser를 선택하고 폼으로 마우스를 끌고가서 놓는다. 그러면 Edit Connections대화칸이 나타난다. beforeUpdate()신호를 beforeUpdateBook()처리부에 련결한다. beforeInsert()신호를 beforeUpdateBook()처리부에 련결한다. 끝으로 primeUpdate()신호를 primeUpdateBook()처리부에 련결한다.

③ 기본코드를 작성한다. 코드는 qt/tools/designer/examples/book/book7/editbook.ui에서 발취하였다. 다음과 같이 조작한다.

init()함수로 시작한다. 이 함수는 대화칸이 구성된 다음에 호출되고 그것을 사용하여 저자이름을 ComboBox에 삽입한다.

```
void EditBookForm::init()
{
    QSqlQuery query( "SELECT surname FROM author ORDER BY surname;" );
    while ( query.next() )
        ComboBoxAuthor->insertItem( query.value( 0 ).toString());
}
```

여기에서는 질문을 실행하여 저자이름목록을 얻고 ComboBox에 각각 삽입한다.

다음에 레코드가 자료기지에서 갱신(삽입)되기전에 실행되는 코드를 쓴다.

```
void EditBookForm::beforeUpdateBook( QSqlRecord * buffer )
{
    QSqlQuery query( "SELECT id FROM author WHERE surname ='" +
        ComboBoxAuthor->currentText() + "';" );
    if ( query.next() )
        buffer->setValue( "authorid", query.value( 0 ) );
}
```

ComboBox에서 현재저자의 id를 찾고 그것을 갱신(또는 삽입)완충기의 authorid 마당에 배치한다.

사용자가 레코드를 항행할 때 ComboBox가 현재의 저자를 반영한다는것을 담보한다.

```
void EditBookForm::primeUpdateBook( QSqlRecord * buffer )
{
    // Who is this book's author?
    QSqlQuery query( "SELECT surname FROM author WHERE id='" +
        buffer->value( "authorid" ).toString() + "';" );
    QString author = "";
    if ( query.next() )
        author = query.value( 0 ).toString();
    // Set the ComboBox to the right author
    for ( int i = 0; i < ComboBoxAuthor->count(); i++ ) {
        if ( ComboBoxAuthor->text( i ) == author ) {
            ComboBoxAuthor->setCurrentItem( i ) ;
            break;
        }
```

```
    }
  }
```

우선 책의 저자를 찾고 다음으로 그 저자를 찾을 때까지 ComboBox의 항목들을 순환하여 대조되는 저자로 ComboBox의 현재항목을 설정한다.

저자이름이 바뀌거나 삭제되였으면 질문은 실패하고 저자id는 완충기에 삽입되지 않는다. ComboBox를 구성할 때 저자id들을 기록하거나 QMap에 보관하고 필요에 따라서 찾는 방법을 선택한다. 이 수법은 init()와 beforeUpdateBook(), primeInsertBook()함수에 대한 변경과 새로운 함수 mapAuthor()의 추가를 요구한다. qt/tools/designer/examples/book/book8/editbook.ui로부터 련관된 코드를 아래에 보여준다.

① 우선 저자이름을 저자id로 넘기기 위하여 클라스변수를 창조한다. Object Explorer의 Members타브를 선택하고 Class Variables항목을 오른쪽 단추로 선택하고 New를 찰칵한다. QMap<QString,int> authorMap라고 입력한다.

② 그다음 저자id들을 init()함수에 기록한다.

```
void EditBookForm::init()
{
  QSqlQuery query("SELECT surname, id FROM author ORDER BY surname;);
  while ( query.next() ) {
    ComboBoxAuthor->insertItem( query.value( 0 ).toString() );
    int id = query.value( 1 ).toInt();
    mapAuthor( query.value( 0 ).toString(), id, TRUE );
  }
}
```

매 저자의 이름을 ComboBox에 삽입한 다음에 저자이름과 id로서 QMap를 구성한다.

③ 저자id를 자료기지에서 조사하지 않고 QMap에서 조사한다.

```
void EditBookForm::beforeUpdateBook( QSqlRecord * buffer )
{
  int id;
  mapAuthor( ComboBoxAuthor->currentText(), id, FALSE );
  buffer->setValue( "authorid", id );
}
```

④ 저자id들을 보관하고 돌려주는 하나의 함수를 사용하여 정적자료구조를 사용할 수 있게 한다.

```
void EditBookForm::mapAuthor(const QString & name, int & id, bool populate)
{
```

```
    if ( populate )
        authorMap[ name ] = id;
    else
        id = authorMap[ name ];
}
```

populate기발이 TRUE이면 저자의 이름과 id를 QMap에 보관하고 그렇지 않으면 주어진 저자이름을 찾고 id를 적당히 설정한다.

⑤ 갱신하기 전에 author복합칸이 정확한 저자를 보여준다는것을 담보해야 한다.

```
void EditBookForm::primeUpdateBook( QSqlRecord * buffer )
{
    int id = buffer->value( "authorid" ).toInt();
    for ( int i = 0; i < ComboBoxAuthor->count(); i++ ) {
        QString author = ComboBoxAuthor->text( i );
        if ( authorMap.contains( author ) && authorMap[author] == id ) {
            ComboBoxAuthor->setCurrentItem( i ) ;
            break;
        }
    }
}
```

응용프로그람의 각이한 부분들에서 같은 외부열쇠검색이 요구될 때 특별히 유용한 다른 수법은 유표의 파생클라스를 작성하고 이것을 검색에 사용하는것이다.

제8절. Qt Designer의 전용화와 통합

1. Qt Designer의 전용화

Qt Designer는 두가지 방법으로 전용화할수 있으며 사용자정의창문부품을 추가하여 Qt Designer의 작업환경을 변경할수 있다. 사용자정의창문부품은 6절에서 설명하였다. 여기서는 Qt Designer자체의 전용화에 초점을 둔다.

Qt Designer의 도구띠는 모두 류동가능하므로 도구띠핸들에 의하여 끌고다니고 마음대로 배렬할수 있다. 또한 Files와 Object Hierarchy, Property Editor, Output Windows은 류동가능하므로 자기가 요구하는 위치까지 끌고갈수 있다. 또한 Qt Designer의 류동령역밖으로 끌고감으로써 류동창문으로 만들수 있다.

Edit|Preferences를 선택하여 Preferences대화칸을 열고 일반적인 선택을 설정할수 있다. Restore Last Workspace on Startup검사칸을 선택하면 Qt Designer는 도

구띠와 류동가능창문의 크기와 위치를 기억한다. 색이나 픽스매프를 선택하여 Qt Designer의 기본창문배경을 변경할수 있다. 또한 배치의 사용이 여분의 살창을 만드므로 살창의 표시를 해제할수 있다.

Preferences대화칸은 설치한 플라그인에 따라서 추가적인 타브를 가질수 있다. C++ Editor타브가 기정으로 설치되므로 이것을 설명한다.

C++ Editor타브는 Qt Designer코드편집기에서 강조표시되는 구문용서체를 선택하는데 사용된다. 모든 원소에 해당한 기준서체는 목록의 마지막 항목인 Standard원소로 설정된다. 처음부터 끝까지 하나의 서체를 사용하려고 Standard서체를 설정하면 다른 모든 원소들은 그 설정을 계승한다.

2. Qt Designer의 코드편집기

코드편집기는 Editor플라그인이 설치되면 사용할수 있다. C++ Editor플라그인은 기정으로 설치된다.

코드편집기는 다음의 건결합을 제공한다.

Left Arrow -- 유표를 한 문자 왼쪽으로 이동한다.

Right Arrow -- 유표를 한 문자 오른쪽으로 이동한다.

Up Arrow -- 유표를 한행 우로 이동한다.

Down Arrow -- 유표를 한행 아래로 이동한다.

Page Up -- 유표를 한 페지 우로 이동한다.

Page Down -- 유표를 한 페지 아래로 이동한다.

Backspace -- 유표의 왼쪽문자를 삭제한다.

Home -- 유표를 행의 선두로 이동한다.

End -- 유표를 행의 끝으로 이동한다.

Delete -- 유표의 오른쪽문자를 삭제한다.

Ctrl+A -- 유표를 행의 선두로 이동한다.

Ctrl+B -- 유표를 한 문자 왼쪽으로 이동한다.

Ctrl+C -- 선택된 본문을 오려둠판에 복사한다. (Windows에서는 Ctrl+Insert)

Ctrl+D -- 유표의 오른쪽문자를 삭제한다.

Ctrl+E -- 유표를 행의 끝으로 이동한다.

Ctrl+F -- Find Text대화칸을 연다.

Ctrl+G -- Goto Line대화칸을 연다.

Ctrl+H -- 유표의 왼쪽문자를 삭제한다.

Ctrl+I -- 행이나 유표를 포함하는 선택본문을 들여쓴다.

Alt+I -- 증분탐색을 시작한다.(아래를 보시오.)

Ctrl+K -- 유표위치부터 행끝까지 삭제한다.

Ctrl+N -- 유표를 한행 아래로 이동한다.

Ctrl+P -- 유표를 한행 우로 이동한다.

Ctrl+R -- Replace Text대화칸을 연다.

Ctrl+V -- 오려둠판의 본문을 행편집에 붙이기한다. (Windows에서는 Shift+Insert)

Ctrl+X -- 표식된 본문을 자르기하여 오려둠판에 복사한다.(또한 Windows에서는 Shift+Delete)

Ctrl+Y -- 마지막 조작을 재실행한다.

Ctrl+Z -- 마지막 조작을 취소한다.

Ctrl+Left Arrow -- 유표를 한 단어 왼쪽으로 이동한다.

Ctrl+Right Arrow -- 유표를 한 단어 오른쪽으로 이동한다.

Ctrl+Up Arrow -- 유표를 한 단어 우로 이동한다.

Ctrl+Down Arrow -- 유표를 한 단어 아래로 이동한다.

Ctrl+Home Arrow -- 유표를 본문의 선두로 이동한다.

Ctrl+End Arrow -- 유표를 본문의 끝으로 이동한다.

Tab - 보완.

본문을 선택(표식)하기 위하여 Shift건을 누르면서 이동건중 하나를 누른다. 례를 들면 Shift+Right Arrow는 오른쪽 문자를 선택하고 Shift+Ctrl+Right Arrow는 오른쪽 단어를 선택한다.

Alt+I를 눌러서 증분탐색을 시작한다. 입력한 문자들은 Search도구띠의 Incremental Search행편집에 나타나고 유표는 편집기에서 처음으로 일치하는 본문에로 이동된다. 입력하면 탐색을 계속한다. Return을 눌러서 다음의 일치하는 본문으로 이동하고 Esc를 눌러서 그때 도달한 위치에서 탐색을 취소한다.

1개이상의 문자를 입력한 다음 Tab를 눌러서 보완한다. 보완작업을 다음과 같다. 본문입력을 시작하고 Tab를 누른다. 편집기가 같은 문자들로 시작하는 본문의 다른 항목을 발견한다면 본문을 보완하게 한다. 가능한 본문을 한개이상 발견하면 선택목록을 펼친다. 화살건을 사용하여 본문을 하나 선택하고 Return을 누르거나 Esc를 눌러서 입력을 계속한다. Preferences대화칸에서 보완을 차단할수 있다.

-> 혹은 .을 입력할 때 편집기는 지령보완목록을 펼친다. 이때 화살건을 사용하여 요구하는 항목으로 이동하고 Return를 누르거나 Esc를 눌러서 목록을 무시한다.

3. 형판의 작성과 리용

Qt Designer는 두가지 수법으로 형판폼을 창조한다. 가장 간단한 수법은 형판등록부에 .ui파일을 보관하는것이다. 둘째 수법은 형판을 사용하는 폼들에서 기초클라스로 사용하려는 용기창문부품클라스의 창조한다. 여기서는 두가지 기술을 설명한다.

1) 간단한 형판

이 형판들은 공통창문부품들을 가지는 폼들의 전체 모임을 창조하려고 할 때 가장 효과적이다. 실례로 프로젝트가 많은 폼들을 요구하고 그 모두가 회사이름과 략호(logo)가 있는 상표를 요구한다.

우선 간단한 형판을 창조한다.

① File|New를 선택하여 New File대화칸을 연다. Dialog template를 선택하고 OK를 찰칵한다.

② Text Label도구띠단추를 찰칵하고 폼의 왼쪽웃구석근방을 선택한다. font Point Size속성을 16으로, text속성을 자기 또는 자기 회사의 이름으로 변경한다. Line 도구띠단추를 찰칵하고 표식자아래의 폼부분을 선택한다. 그리고 튀여나오기차림표에서 Horizontal을 선택한다.

③ 표식자과 행을 선택한다.(Ctrl을 누르면서 폼을 선택하고 선택창을 행과 표식자가 닿거나 포함되도록 확대한다.) Ctrl+L을 눌러서 수직으로 배치한다.

④ Save도구띠단추를 찰칵한다. Save As대화칸에서 Qt Designer의 형판등록부로 이행한다. 실례로 qt/tools/designer/templates를 들수 있다. 이름을 Simple_Dialog.ui라고 입력하고 Save를 선택한다.

⑤ Forms목록에서 폼을 오른쪽 단추로 찰칵하고 Remove form from project를 선택한다.

간단한 형판을 가지게 되고 그것을 사용할 준비가 되였다. File|New를 선택하고 New File대화칸을 선택한다. 나타나는 형판들중 하나는 Simple Dialog이다. 단순한 대화칸을 선택하고 OK를 찰칵한다. 새로운 폼이 형판와 같은 창문부품과 배치와 함께 나타난다. 다른 창문부품과 기능을 추가한다. 폼을 보관하려고 시도할 때 새로운 폼의 이름을 묻는다.

2) 기초클라스형판

이 형판들은 기초클라스에 기초한 모든 폼들이 계승할수 있는 기정기능을 제공하려고 할 때 쓸모있다. 실례에서는 형판의 기초로서 크기를 보관하는 SizeAware라는 클라스를 사용한다. 클라스자체를 서술하지 않지만 Qt Designer형판로서 그것을 사용하게 하는데 초점을 둔다. 이 클라스의 원천은 qt/tools/designer/examples/sizeaware에 있다.

형판는 사용자정의창문부품이나 현존용기창문부품에 기초할수 있다.

형판이 사용자정의창문부품에 기초하게 하려면 우선 그것을 Qt Designer의 사용자정의창문부품에 추가하여야 한다. Tools|Custom|Edit Custom Widgets를 선택하여 Edit Custom Widgets대화칸을 연다(6절 소절1 참고). New Widget를 선택한다. 클라스를 MyCustomWidget로부터 SizeAware로 변경한다. Header file생략단추를 찰칵하고 파일 qt/tools/designer/examples/sizeaware/sizeaware.h을 선택한다. Container Widget검사

칸을 선택한다. 이 클라스는 2가지 속성을 제공한다. Properties타브를 선택한다. New Property를 선택하고 속성이름을 company로 변경한다. 다시 New Property를 선택하고 속성이름을 settingsFile로 변경하고 Close를 선택한다.

현존창문부품 혹은 자체의 사용자정의창문부품에 기초하여 형판을 창조하려면 File|Create Template를 선택하여 Create Template대화칸을 연다. Template Name을 SizeAware로 변경하고 SizeAware기초클라스를 선택하고 Create를 찰칵한다. 대화칸은 형판을 창조하고 즉시 그자체를 닫는다. Qt Designer를 닫고 그것을 재기동한다.

새 형판 SizeAware는 현재 형판목록에서 사용할수 있다. File|New를 선택하고 SizeAware를 선택하고 OK를 찰칵한다. 두개의 속성 company와 settingsFile은 Properties창문에서 사용할수 있다. 이 형판에 기초하는 폼들은 자기의 크기를 기억하며 재적재될 때 크기가 조절된다.(실천에서 폼마다 하나의 settingsFile을 가지는 응용프로그람을 권고하지 않으므로 이 형판는 실제로 단일한 기본창문을 가지는 응용프로그람에만 사용할수 있다.)

4. Qt Designer와 Visual Studio의 통합

Qt Designer는 Qt에 제공되는 theqmsdev.dll파일을 사용하여 Visual Studio 6.0에 통합할수 있다. Visual Studio.Net는 이러한 통합을 지원하지 않는다.

새로운 도구띠가 Visual Studio에 나타난다. 도구띠단추들은 다음과 같다.

New Qt Project -- 작은 응용프로그람위자드

New Qt Dialog -- 빈 Qt Dialog를 현재 작업중인 프로젝트에 추가하거나 현존 대화칸을 추가한다.

Qt GUI Designer -- Qt Designer를 실행한다.

Open Qt Project -- .pro를 가지고 qmake를 실행한다.

Write Qt Project -- 현재의 VS프로젝트를 .pro파일로서 보관한다.

Use Qt -- Qt서고들을 현재 작업중인 프로젝트에 추가한다.

Add MOC -- 현재 작업파일에 moc사전콤파일러를 추가한다.

작업공간창문에서 .ui파일을 두번 련속 누르면 Qt Designer가 호출된다.

Q_OBJECT마크로를 포함하는 .cpp파일을 창조한다면 moc에 의해 생성되는 추가적인 파일이 프로젝트에 포함되여야 한다. 실례로 file.cpp가 있다면 마지막 행이 #include "file.moc"이여야 하며 추가파일이 file.moc를 호출한다. 그것을 담보하기 위하여 Visual Studio가 moc를 실행하여 이 파일을 생성하고 전용의존관계를 생성한다. 프로젝트작업공간에서 Q_OBJECT마크로를 포함하는 .cpp파일을 두번 련속 찰칵한다. Add MOC도구띠단추를 찰칵하여 프로젝트작업공간에서 빈 .moc파일을 생성한다. 새로 생성한 .moc파일을 오른쪽 단추로 찰칵하고 올리펼침차림표에서 Settings를 선택하여 Project Settings대화칸을 펼친다. Custom Build타브를 찰칵한다. Dependencies단추를 찰칵

하여 User Defined Dependencies대화칸을 펼친다. $(InputDir)\$(InputPath)를 입력한 다음 Return을 누른다. OK를 찰칵하여 Dependencies대화칸을 닫은 다음 OK를 찰칵하여 Project Settings대화칸을 닫는다.

그 add-in을 삭제하려면 도구띠에서 그것을 삭제하고 add-ins등록부에서 theqmsdev.dll파일을 삭제한다.

1) qmake없이 Makefile작성

Qt에 제공된 qmake도구는 .pro프로젝트파일에 기초하여 가동환경에 적당한 Makefile을 창조할수 있다. 여기서는 Qt응용프로그람의 건설에 포함된 의존관계를 서술하며 한쌍의 간단한 실례Makefile을 준다. 그리고 Makefile에 대하여 리해하고있는 것으로 가정한다.

Qt Designer는 콤파일러가 콤파일하는 .h와 .cpp파일들을 생성하는데 사용되는 .ui파일을 생성한다. .ui파일들은 uic에 의하여 처리된다. QObject로부터 계승되는 클라스들 실례로 처리부와 신호들을 사용하는 클라스들은 추가적인 .cpp파일의 생성을 요구한다. 이 파일들은 moc에 의해 생성되는데 원래의 .cpp파일이 file.cpp로 불리울 때 moc_file.cpp로 이름지어진다. .cpp파일에 Q_OBJECT마크로가 포함되여있으면 추가적인 파일 file.moc가 생성되여야 하며 .cpp에서 보통 끝에 포함되여야 한다. 이것은 여분의 의존관계창조를 요구한다.

uic에 의한 .ui파일처리는 두번에 걸쳐 수행된다.

```
uic myform.ui -o myform.h
uic myform.ui -i myform.h -o myform.cpp
```

첫째 실행은 머리부파일을 창조하고 두번째 실행은 .cpp파일을 창조한다. 폼의 파생클라스를 만들려고 한다면 uic를 사용하여 파생클라스골격을 생성할수 있다.

```
uic formbase.ui -o formbase.h
uic formbase.ui -i formbase.h -o formbase.cpp
uic -subdecl Form formbase.h formbase.ui -o form.h
uic -subimpl Form form.h formbase.ui -o form.cpp
```

우선 기초클라스를 위한 머리부파일과 실현파일을 생성한다. 그다음 파생클라스를 위한 머리부파일과 실현파일골격을 생성한다. 골격생성에서 uic의 사용은 Makefile에서 아무것도 수행하지 않는다. 또한 지령행에서 -subdecl과 -subimpl은 좀 다르다.

QObject로부터 계승되는 클라스들을 포함하는 실현파일들에 대하여 moc파일들을 창조해야 한다.

```
moc myform.h -o moc_myform.cpp
```

간단한 Makefile을 통하여 실제로 의존관계를 고찰하자.

```
myapp: moc_myform.o myform.o main.o
```

```
        g++ -lqt -o myapp moc_myform.o myform.o main.o
    main.o: main.cpp
        g++ -o main.o main.cpp
    moc_myform.o: moc_myform.cpp
        g++ -o moc_myform.o moc_myform.cpp
    moc_myform.cpp: myform.h
        moc myform.h -o moc_myform.cpp
    myform.o: myform.cpp
        g++ -o myform.o myform.cpp
    myform.cpp: myform.h myform.ui
        uic myform.ui -i myform.h -o myform.cpp
    myform.h: myform.ui
        uic myform.ui -o myform.h
```

Makefile에서 완전지령경로를 포함해야 하며 Windows에서 파일이름은 moc.exe와 uic.exe이다.

Unix/Linux환경에서 make지령은 더 많은것을 수행할수 있으므로 다음과 같이 더 간단한 Makefile을 사용할수 있어야 한다.

```
    myapp: moc_myform.o myform.o main.o
        g++ -lq -o $@ $^
    %.o: %.cpp
        g++ -o $^ $@
    moc_%.cpp: %.h
        moc $^ -o $@
    myform.cpp: myform.h myform.ui
        uic myform.ui -i myform.h -o myform.cpp
    myform.h: myform.ui
        uic myform.ui -o myform.h
```

복잡한 Makefile을 고찰하려면 단지 임의의 Qt프로젝트에 대하여 또는 Qt에 제공된 실례에 대하여 qmake를 사용하여 생성한다.

5. 다른 파일형식의 반입

외부파일형식으로 파일을 반입하기 위하여서는 File|Open을 누른 다음 File Type 복합칸을 선택하고 적재하려는 파일형을 고른다. 필요한 파일을 선택하면 Qt Designer는 그 파일을 변환하여 적재한다.

Qt Designer가 외부파일형식을 읽는데 사용하는 려파기는 진척중에 있다. 여기서

서술하는것보다 자기의 Qt Designer판에서 사용할수 있는 다른 려파기를 가질수 있다. 유효려파기를 보는 가장 간단한 방법은 파일열기대화칸을 열어보는것인데 모든 려파기가 File Type복합칸에 표시된다.

1) Qt Architect파일들의 반입

Qt Architect는 Jeff Harris와 Klaus Ebner가 작성한 Qt용의 무료GUI건설프로그람이다. .dlg확장자는 Qt Architect대화칸파일과 련관된다.

Qt Designer는 Qt Architect 2.1이상에 의하여 생성된 파일들을 읽을수 있다. 이전판의 Qt Architect로부터 .dlg파일이 주어지면 Qt Designer는 그것을 2.1판의 파일형식으로 변환하는 방법을 말해준다. (변환절차는 .dlg파일의 판에 따라 변한다.)

반입려파기는 .dlg파일반입작업을 수행하며 결과는 Qt Architect에서 얻은것과 거의 같다. 그러나 대화칸을 사용하는 C++코드는 약간한 수정을 요구한다.

도구들사이의 차이로 인하여 Qt Architect파일들을 Qt Designer형식으로 변환하는데 다음과 같은 몇가지 결함이 있다.

• 공간과 여백의 배치 - .dlg파일배치가 공간과 여백의 배치에 Qt Architect기정값을 사용하면 Qt Designer는 이것들을 무시하고 자기의 표준기정값을 사용한다. 필요하다면 layoutSpacing과 layoutMargin속성을 수동으로 변경할수 있다.

• 배치확대와 공간 - Qt Architect는 Qt Designer보다도 Qt배치체계의 더 많은 특성(즉 확대와 공간)에로의 호출을 준다. Qt Designer는 이 특성을 사용하는 .dlg파일들에 잘 대항하려고 시도하지만 때때로 크기조절은 바라는것이 아니다. 일반적으로 해결책에는 창문부품들의 sizePolicy속성의 설정과 수축자의 삽입 혹은 삭제가 포함된다.

• 관리 및 비관리창문부품들의 혼합 - Qt Architect는 창문부품이 배치에 의하여 관리되는 자식창문부품들과 고정위치를 가지는 다른 자식창문부품들을 가지게 한다. 여기에 사용하는 .dlg파일이 주어질 때 Qt Designer는 조용히 고정위치창문부품들을 그 배치에 넣는다.

• 픽스매프 - Qt Designer는 .dlg파일들에 지정된 픽스매프들을 무시한다. 이것들은 Qt Designer에서 수동적으로 되살려야 한다.

2) Glade파일들의 반입

Glade는 Damon Chaplin이 작성한 GTK+와 GNOME용의 자유GUI건설프로그람이다. 확장자 .glade는 Glade파일들을 의미한다.

Qt Designer는 판본 0.6.0까지 Glade파일들로 시험하였으며 물론 그후판과도 작업할수 있다.

Glade가 Qt를 목표로 하지 않아도 배치체계와 GTK+의 창문부품모임은 Qt와 류사하므로 려파기는 .glade파일에서 대부분의 정보를 얻는다.

Glade파일들의 변환과 관련하여 다음과 같이 고려할것이 있다.

• 표식자안의 & - Qt는 QLabel이 짝패가 아닐 때 &를 표시한다.(짝패는 QLabel을 대신하여 초점을 받아들이는 창문부품이다.) Glade는 (밑줄이 있는)지름건을 가지는 GtkLabel창문부품을 허용하지만 어떤 짝패도 가지지 않는다. 이것은 사용자가 밑줄문자가 지름건일것을 기대하므로 오유로 된다. 이 상황에서 Qt는 지름건을 밑줄로 표시하지 않고 &자체를 표시한다. 이때에는 QLabel창문부품들을 조사하고 buddy속성을 설정해야 한다.

• 대리기호(placeholder)배치 - GTK는 배치위치를 대리기호가 차지하도록 한다. Qt Designer는 이 대리기호들을 본문이 붉은색 QLabel들로 변환되게 하므로 그것들을 발견하여 수동적으로 고착시킬수 있다.

• Qt와 등가하지 않은 GTK+ 혹은 GNOME창문부품 - Qt는 대부분의 GTK+창문부품들과 등가하지만 Glade도 역시 GNOME을 지원하며 그 목표는 완전한 탁상환경을 제공하는것이다. Qt의 유효범위가 협소하므로 Qt Designer는 변환할수 없는 창문부품과 조우할 때 그것을 문제를 가리키는 표식자로 교체한다. 실례로 GnomePaperSelector는 본문이 붉은색의 GnomePaperSelector?인 QLabel로 교체된다. KDE에 이식하려고 한다면 대응하는 KDE창문부품을 사용하려고 할수 있다.

다른 GTK+/GNOME창문부품들은 오직 일정한 상황에서만 유지된다. 실례로 GnomeDruid는 다른 창문부품에 매몰될수 있으며 한편 대응하는 QWizard클라스는 매몰될수 없다.

• 통보창문과 다른 고준위대화칸 - Glade는 GnomeMessageBox와 GtkFileSelection, GtkFontSelectionDialog 등의 편집을 유지한다. 이것은 보통 Qt에서 QMessageBox대화칸와 QFileDialog, QFontDialog 등에 의하여 C++코드로 얻어진다.

• 독자적인 튀여나오기차림표 - Qt Designer는 오직 QMainWindow안에서 튀여나오기차림표를 유지한다. 독자적인 튀여나오기차림표(가령 문맥차림표)를 요구한다면 QPopupMenu를 사용하여 이것을 수행하는 코드를 간단히 쓸수 있다.

• 크기방략파라메터 - Glade는 속성편집기의 Place타브에서 크기방략을 제공한다. Qt기정값들이 보통 충분히 좋을 때 Qt Designer는 padding과 expand, shrink, fill정보를 사용하려고 하지 않는다. 일부 경우에 sizePolicy속성을 수동적으로 설정하여 요구하는 효과를 얻을수 있다.

• GNOME표준그림기호 - GNOME는 아주 큰 표준그림기호모임을 제공한다. Qt Designer는 이것들에 대한 참고를 무시한다. KDE에 이식한다면 수동적으로 표준KDE그림기호들을 설정해야 한다.

• Packer배치 - GTK+는 다른 종류의 배치에 쓰이는 GtkPacker라는 클라스를 제공한다. Qt는 QPackerLayout를 제공하지 않는다. Qt Designer는 packer배치를 수직배

치처럼 취급하며 그것이 정확한 효과를 생성하도록 배치들의 결합으로 변경해야 한다.

• 변환후 부정확하다고 인정된 본문 - hAlign속성은 때때로 잘못 설정된다. 이 경우에 그것을 수동으로 변경해야 한다. Glade의 변화로 발생된다.

제9절. 지름건

Ctrl+A - 작업중인 폼안의 GUI원소들을 모두 선택한다.

Ctrl+B - 선택된 배치를 파괴하여 GUI원소를 추가, 삭제할수 있게 한다.

Ctrl+C - 작업중인 폼으로부터 오려둠판에 선택한 GUI원소들을 복사한다.

Alt+E - Edit차림표를 펼친다.

Alt+F - File차림표를 펼친다.

Ctrl+G - 선택된 용기에 살창배치를 적용하거나 선택된 GUI원소들을 포함하는 새 용기를 창조하고 살창배치를 이 용기에 적용한다.

Ctrl+H - 선택된 용기에 수평칸배치를 적용하거나 선택된 GUI원소들을 포함하는 새 용기를 창조하고 수평칸배치를 이 용기에 적용한다.

Alt+H - Help차림표를 펼친다.

Ctrl+J - 선택된 GUI원소(또는 원소들)의 크기를 조절하여 그자체를 적당히 표시하는데 필요한 최소크기를 가지게 한다.

Ctrl+L - 선택된 용기에 수직칸배치를 적용하거나 선택된 GUI원소들을 포함하는 새 용기를 창조하고 수직칸배치를 이 용기에 적용한다.

Alt+L - Layout차림표를 펼친다.

Ctrl+M - Qt Assistant에서 직결방조문서를 연다.

Ctrl+N - New File대화칸을 연다.

Ctrl+O - Open File대화칸을 연다.

Alt+P - Preview차림표를 펼친다.

Ctrl+R - 작업중에 있는 폼에서 지름건들이 중복되는가 검사한다.

Ctrl+S - 작업중에 있는 폼을 보관한다.

Ctrl+T - 가동환경의 기정GUI형식으로 작업중에 있는 폼을 미리보기한다.

Alt+T - Tools차림표를 펼친다.

Ctrl+V - 오려둠판의 GUI원소를 작업중에 있는 폼에서 원래의 폼에 있던 위치로부터 약간 변위하여 붙이기한다. 오려둠판에 GUI원소가 없다면 아무일도 하지 않는다.

Alt+W - Window차림표를 펼친다.

Ctrl+X - 선택된 GUI원소를 작업중에 있는 폼에서 자르기하여 오려둠판에 넣는다.

Ctrl+Y - 마지막 취소동작을 재시행한다.

Ctrl+Z - 마지막 동작을 취소한다.

Del - 선택된 GUI원소들을 작업중에 있는 폼에서 삭제한다.

F1 - Qt Assistant에서 Qt Designer지도서의 소개페지를 연다.

Shift+F1 - What's This방식을 투입한다. 이것은 Qt Designer에서 GUI원소를 선택하여 이 원소에 대한 작은 설명창문을 얻게 한다.

F2 - GUI원소들을 선택하게 하는 지시자도구가 동작하게 한다.

F3 - 폼안의 신호-처리부련결을 편집하는 련결도구가 동작하게 한다.

F4 - 작업폼우의 GUI원소들의 타브순서를 변경하는 타브순서도구가 동작하게 한다.

Ctrl+F4 - 작업중인 창문을 닫는다.

Ctrl+F6 - 다음번에 창조한 창문을 작업창문으로 한다.

Ctrl+Shift-F6 - 다음번에 창조한 창문을 작업창문으로 한다.

제10절. 차림표선택

Qt Designer는 응용프로그람을 창조하는데 필요한 작용을 호출하는 차림표선택을 제공한다. 대부분의 차림표선택은 추가적인 선택과 기능을 제공하는 대화칸창문을 펼치게 한다. 가장 일반적으로 사용되는 차림표선택들은 또한 대응하는 도구띠단추들을 가지고있다. 이 절에서는 각 차림표선택과 그 사용법을 설명한다.

1. File차림표

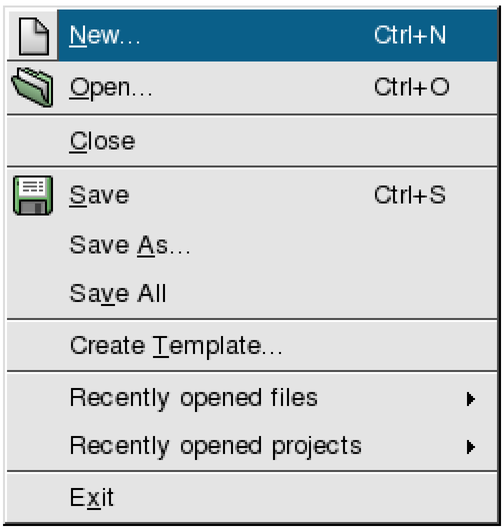


그림 1-70. File차림표

File차림표는 Alt+F에 의해 호출되며 다음과 같은 차림표선택을 제공한다.

File|New을 선택하여(또는 Ctrl+N을 눌러서) 새 프로젝트, 폼 혹은 파일을 창조

한다. 이 선택은 New File대화칸을 호출한다.

File|Open을 선택하여 (또는 Ctrl+O을 눌러서) 현존 프로젝트, 폼 혹은 파일을 연다. 파일이름을 선택할수 있는 File Open대화칸이 열린다.

File|Close를 선택하여 현재 열린 프로젝트를 닫는다. 프로젝트에 보관하지 못한 변경이 있으면 Save Form대화칸이 표시된다.

File|Save를 선택하여 (또는 Ctrl+S을 눌러서) 프로젝트를 그 폼과 파일들과 함께 보관한다. 폼이나 파일들을 가지는 프로젝트에 대하여 Save를 눌러서 프로젝트를 완료하기전에 보관한다. 새 폼들에 대하여 Save를 선택하면 Save Form As Dialog가 나타난다. 이전에 보관하였던 폼들에 대해서는 Save를 선택한다. 새 파일들이나 변경된 파일들에 대하여서는 Save를 찰칵한다.

File|Save As를 선택하여 현재폼이나 파일을 보관하고 이름을 짓는다. 이 선택은 Save Form As Dialog를 펼친다.

File|Save All을 선택하여 매개의 열린 프로젝트에 매개의 열린파일과 폼을 보관한다.

File|Create Template를 선택하여 폼형판을 창조한다. 이 선택은 Create Template Dialog대화칸을 펼친다.

File|Recently Opened Files를 선택하여 제일 최근에 열었던 파일들을 표시한다. 그 파일들중 하나를 선택하여 연다. 파일들보다 프로젝트를 열것을 권고한다. 프로젝트의 Project Overview Window에서 파일이름을 선택하여 파일을 열수 있다.

File|Recently Opened Projects를 선택하여 제일 최근에 연 프로젝트들을 표시한다. 표시된 프로젝트들중 하나를 선택하여 연다.

File|Exit를 선택하여 Qt Designer를 완료한다. 어떤 열린 파일들에 보관하지 못한 변경이 있으면 Qt Designer가 완료하기전에 Save Form Dialog통보칸이 그 매개에 대하여 나타난다. 이전에 보관하지 않았으나 변경된 폼에 대하여 혹은 보관했었는데 변경된 폼에 대하여 Save Form Dialog이 호출된다. Yes를 눌러서 Save Form As대화칸을 펼친다.

2. Edit차림표

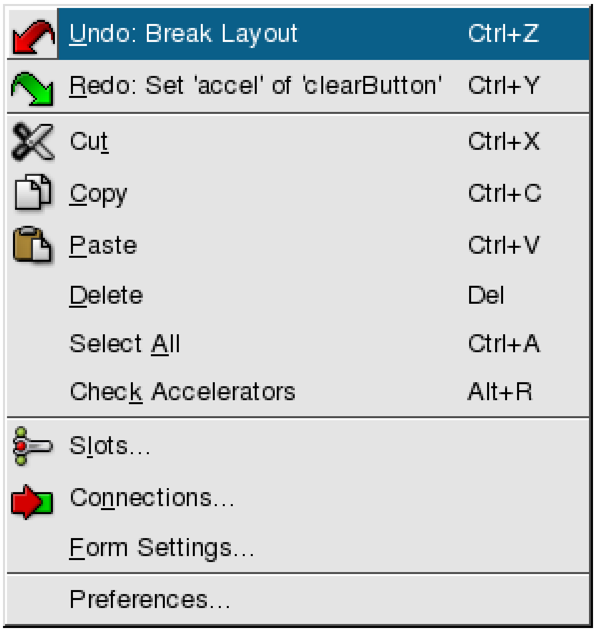


그림 1-71. Edit차림표

Edit차림표는 Alt+E에 의해 호출되며 다음과 같은 차림표선택을 제공한다.

Edit|Undo를 선택하여 (또는 Ctrl+Z를 눌러서) 동작을 취소한다. 수행된 마지막 동작의 이름이 Undo뒤에 나타난다.

Edit|Redo를 선택하여 (또는 Ctrl+Y를 눌러서) 동작을 재시행한다. 수행된 마지막 동작의 이름이 Redo뒤에 나타난다.

Edit|Cut를 선택하여 (또는 Ctrl+X를 눌러서) 현재 폼이나 파일로부터 선택항목을 삭제하고 그것을 오려둠판에 복사한다.

Edit|Copy를 선택하여 (또는 Ctrl+C를 눌러서) 현재 폼이나 파일로부터 선택항목을 오려둠판에 복사한다.

Edit|Paste를 선택하여 (또는 Ctrl+V를 눌러서) 오려둠판의 항목(있다면)을 현재 폼이나 파일에 붙이기한다.

Edit|Delete를 선택하여 (또는 Del을 눌러서) 현재 폼이나 파일로부터 선택항목을 삭제한다.

Edit|Select All을 선택하여 (또는 Ctrl+A를 눌러서) 현재 폼의 모든 창문부품 혹은 현재 파일안의 모든 본문을 강조표시한다.

Edit|Check Accelerators를 선택하여 (또는 Alt+R를 눌러서) 모든 지름건들이 오직 한번만 쓰인다는것을 확인한다. 지름건이 한번이상 사용되면 'The accelerator 'x' is used 'y' times'문의 통보칸이 나타난다. Select를 선택하여 같은 지름건을 가지는 창문부품들을 강조표시하거나 Cancel을 찰칵하여 어떤 동작이 없이 통보칸을 완료한다.

Edit|Slots를 선택하여 처리부와 함수들을 편집하고 창조한다. 이 선택은 Edit Functions대화칸을 펼친다.

Edit|Connections을 선택하여 View and Edit Connections대화칸을 펼친다.

Edit|Form Settings를 선택하여 Form Settings대화칸을 펼친다.

Edit|Preferences를 선택하여 Preferences대화칸을 펼친다.

3. Project차림표

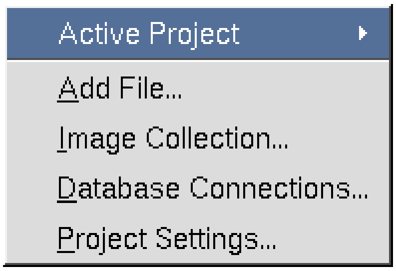


그림 1-72. Project차림표

Project차림표는 Alt+O에 의해 호출되며 다음과 같은 차림표선택을 제공한다.

Project|Active Project를 선택하여 하나이상의 프로젝트가 열려있다면 프로젝트들을 절환할수 있다. 또한 File도구띠단추안의 Active Project내리펼침복합칸을 사용하여 프로젝트들사이를 절환할수 있다.

Project|Add File을 선택하여 Add대화칸을 펼친다

Project|Project Settings를 선택하여 Project Settings대화칸을 펼친다.

Project|Image Collection을 선택하여 Manage Image Collection대화칸을 펼친다.

Project|Database Connections를 선택하여 Edit Database Connections대화칸을 펼친다.

4. Search차림표

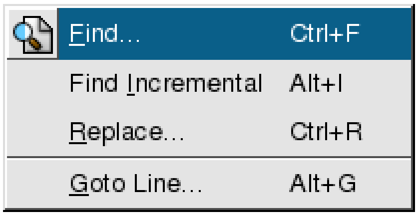


그림 1-73. Search차림표

Search차림표는 Alt+S에 의해 호출되며 다음과 같은 차림표선택을 제공한다.

Search|Find를 선택하여 (또는 Ctrl+F를 눌러서) Find Text 대화칸을 펼친다.

Search|Find Incremental을 선택하여 (또는 Alt+I를 눌러서) Find도구띠단추열에 배치된 본문칸에 유표를 놓는다. 본문칸에 문자들을 입력하면 그때 Qt Designer는

파일에서 발견하는 첫 본문을 강조표시한다. Enter건을 눌러서 본문의 다음 출현으로 이행한다. 일단 찾으려는 단어를 발견하면 Esc건을 눌러서 편집기의 유표를 거기에 배치한다.

Search|Replace를 선택하여 (또는 Ctrl+R를 눌러서) Replace Text대화칸을 펼치고 특정한 단어나 문자들로 교체한다.

Search|Goto line을 선택하여 (또는 Alt+G를 눌러서) Goto Line대화칸을 펼치고 파일안의 지정행으로 이행한다.

5. Tools차림표

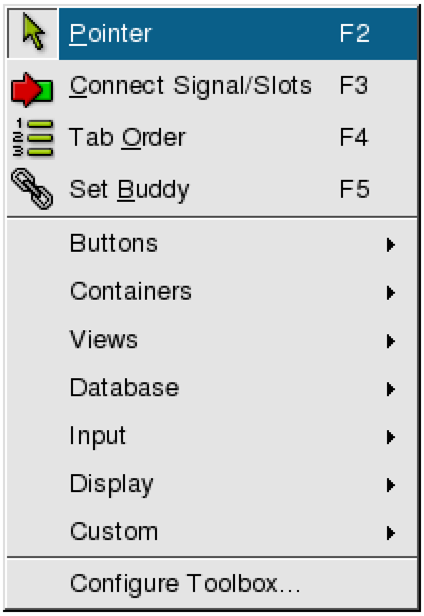


그림 1-74. Tools차림표

Tools차림표는 Alt+T에 의해 호출되며 다음과 같은 차림표선택을 제공한다.

Tools|Pointer을 선택하여(또는 F2를 눌러서) 선택된 창문부품도구띠단추의 선택을 해제한다. 또한 창문부품도구띠단추를 두번 련속 눌렀다면 폼에 새 창문부품삽입을 중지하는데 지적자를 사용한다. Esc건을 눌러서 임의의 시각에 지적자로 돌아온다.

Tools|Connect Signals and Slots을 선택하여(또는 F3을 눌러서) 신호와 처리부를 련결한다. 창문부품을 선택하고 련결을 련결하려는 창문부품(또는 폼)까지 끌기한다. 마우스단추를 놓으면 View and Edit Connections대화칸이 나타난다.

Tools|Tab Order를 선택하여(또는 F4를 눌러서) 건반초점을 받아들일수 있는 폼의 모든 창문부품들에 대하여 타브순서를 설정한다. 이 도구띠단추를 찰칵하고 창문부품들의 곁에 나타나는 수값이 있는 푸른색원을 선택한다. 타브순서의 첫 창문부품을 선택하고 그다음 타브순서의 다음 창문부품을 선택하는 조작을 모든 창문부품들이 요구하는 타브순서번호를 가질 때까지 반복한다. 오유를 범하였으면 첫 창문부품을 두번 련속 누

르고 다시 시작한다. 타브순서방식을 끝내려면 Esc를 누른다. 변경을 취소하려면 타브순서방식을 끝내고 취소한다.

Tools|Set Buddy를 찰칵하여 (또는 F12을 눌러서) 짝패를 표식자로 설정한다. 그다음 표식자를 선택하고 짝패로 하려는 창문부품까지 직선을 끌고간다. 마우스단추를 놓으면 짝패가 설정된다.

Tools|Buttons|PushButton을 선택하고 폼을 선택하여 폼우에 PushButton을 배치한다.

Tools|Buttons|ToolButton을 선택하고 폼을 선택하여 폼우에 ToolButton을 배치한다.

Tools|Buttons|RadioButton을 선택하고 폼을 선택하여 폼우에 RadioButton을 배치한다. ButtonGroup안에 RadioButton들을 배치하여 Qt가 그룹안의 오직 하나의 RadioButton이 한번에 동작상태로 될수 있다는것을 자동적으로 담보한다.

Tools|Buttons|CheckBox를 선택하고 폼을 선택하여 폼우에 CheckBox를 배치한다.

Tools|Containers|GroupBox를 선택하고 폼을 선택하여 폼우에 GroupBox를 배치한다.

Tools|Containers|ButtonGroup을 선택하고 폼을 선택하여 폼우에 ButtonGroup를 배치한다.

Tools|Containers|Frame을 선택하고 폼을 선택하여 폼우에 Frame을 배치한다.

Tools|Containers|TabWidget를 선택하고 폼을 선택하여 폼우에 TabWidget를 배치한다. 타브를 추가 혹은 삭제하려면 타브창문부품을 오른쪽 단추로 찰칵하고 Add Page 혹은 Remove Page를 선택한다.

Tools|Views|ListBox를 선택하고 폼을 선택하여 폼우에 ListBox를 배치한다.

Tools|Views|ListView를 선택하고 폼을 선택하여 폼우에 ListView를 배치한다.

Tools|Views|Icon View를 선택하고 폼을 선택하여 폼우에 IconView를 배치한다.

Tools|Views|Table을 선택하고 폼을 선택하여 폼우에 Table을 배치한다.

Tools|Database|DataTable을 선택하고 폼을 선택하여 폼우에 DataTable을 배치한다.

Tools|Database|DataBrowser를 선택하고 폼을 선택하여 폼우에 DataBrowser를 배치한다.

Tools|Database|DataView를 선택하고 폼을 선택하여 폼우에 DataView를 배치한다.

Tools|Input|LineEdit를 선택하고 폼을 선택하여 폼우에 LineEdit을 배치한다.

Tools|Input|SpinBox를 선택하고 폼을 선택하여 폼우에 SpinBox를 배치한다.

Tools|Input|DateEdit를 선택하고 폼을 선택하여 폼우에 DateEdit를 배치한다.

Tools|Input|TimeEdit를 선택하고 폼을 선택하여 폼우에 TimeEdit를 배치한다.

Tools|Input|DateTimeEdit를 선택하고 폼을 선택하여 폼우에 DateTimeEdit을 배치한다.

Tools|Input|TextEdit를 선택하고 폼을 선택하여 폼우에 TextEdit을 배치한다.

Tools|Input|ComboBox를 선택하고 폼을 선택하여 폼우에 ComboBox을 배치한다.

Tools|Input|Slider를 선택하고 폼을 선택하여 폼우에 Slider을 배치한다.

Tools|Input|ScrollBar를 선택하고 폼을 선택하여 폼우에 Scrollbar을 배치한다.

Tools|Input|Dial을 선택하고 폼을 선택하여 폼우에 Dial을 배치한다.

Tools|Display|TextLabel을 선택하고 폼을 선택하여 폼우에 TextLabel을 배치한다.

Tools|Display|PixmapLabel을 선택하고 폼을 선택하여 폼우에 PixmapLabel을 배치한다.

Tools|Display|LCDNumber를 선택하고 폼을 선택하여 폼우에 LCDNumber을 배치한다.

Tools|Display|Line을 선택하고 폼을 선택하여 폼우에 Line을 배치한다.

Tools|Display|ProgressBar를 선택하고 폼을 선택하여 폼우에 ProgressBar을 배치한다.

Tools|Display|TextBrowser를 선택하고 폼을 선택하여 폼우에 TextBrowser을 배치한다.

Tools|Custom|Edit Custom Widgets를 선택하여 Edit Custom Widgets대화칸을 펼친다.

Tools|Custom|을 선택하고 폼을 선택하여 폼우에 Custom Widget를 배치한다.

Tools|Custom|Edit Custom Widgets를 리용하여 창문부품을 창조한 경우에만 이 차림표선택이 나타난다.

Tools|Configure Toolbox를 선택하여 Configure Toolbox대화칸을 펼친다.

6. Layout차림표

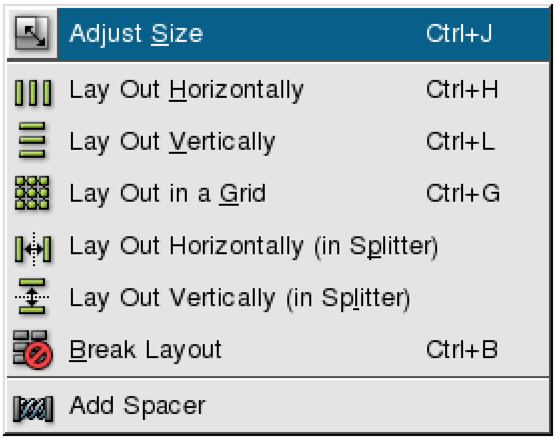


그림 1-75. Layout차림표

Layout차림표는 Alt+L에 의해 호출되며 다음과 같은 차림표선택을 제공한다.

Layout|Adjust Size를 선택하여(또는 Ctrl+J를 눌러서) 권고된 크기로 창문부품의 크기를 조절한다.

Layout|Lay Out Horizontally를 선택하여(또는 Ctrl+H를 눌러서) 선택된 창문부품 혹은 배치들을 배치한다. Shift+Click를 리용하여 각 창문부품 혹은 배치를 선택하고 이 차림표선택을 지정하여 그것들을 수평으로 묶는다. 흔히 복잡한 창문부품들에 대하여 Object Explorer창문의 Widgets타브에서 창문부품과 배치들을 눌러서 그것들을 선택하는것이 가장 간단하다. 오직 한개의 창문부품이 선택되면 그 자식창문부품들이 수평으로 배치된다.

Layout|Lay Out Vertically를 선택하여(또는 Ctrl+L을 눌러서) 선택된 창문부품들을 배치한다. Shift를 누르고 찰칵하여 매개 창문부품 혹은 배치를 선택한 다음 이 차림표항목을 선택하여 그것들을 수직으로 묶는다. 흔히 복잡한 창문부품들에 대하여

Object Explorer창문의 Widgets타브에서 창문부품과 배치들을 눌러서 그것들을 선택하는것이 가장 간단하다. 오직 한개의 창문부품이 선택되면 그 자식창문부품들이 수직으로 배치된다.

Layout|Lay Out in a Grid를 선택하여(또는 Ctrl+G를 눌러서) 선택된 창문부품들을 살창으로 배치한다. 오직 한개의 창문부품이 선택되면 그 자식창문부품들이 살창으로 배치된다.

Layout|Lay Out Horizontally (in Splitter)를 선택하여 선택된 창문부품이나 배치들의 매개사이에 분할기(splitter)를 주어서 배치한다. Shift를 누르고 찰칵하여 각 창문부품이나 배치를 선택하고 이 차림표항목을 선택하여 그것들을 수평으로 묶는다. 흔히 복잡한 창문부품들에 대하여 Object Explorer창문의 Widgets타브에서 창문부품과 배치들을 눌러서 그것들을 선택하는것이 가장 간단하다.

Layout|Lay Out Vertically (in Splitter)를 선택하여 선택된 창문부품이나 배치들의 매개사이에 분할기를 주어서 배치한다. Shift를 누르고 찰칵하여 각 창문부품이나 배치를 선택하고 이 차림표항목을 선택하여 그것들을 수직으로 묶는다. 흔히 복잡한 창문부품들에 대하여 Object Explorer창문의 Widgets타브에서 창문부품과 배치들을 눌러서 그것들을 선택하는것이 가장 간단하다.

Layout|Break Layout를 선택하여 (또는 Ctrl+B를 눌러서) 배치를 파괴한다. 배치를 누르고 이 항목을 선택하면 배치가 삭제된다.

Layout|Add Spacer를 선택하여 폼에 너무 많은 공간을 가지는 창문부품들에 수직 혹은 수평수축자를 추가한다. 수축자는 배치에서 여유공간을 소비한다.

7. Preview차림표

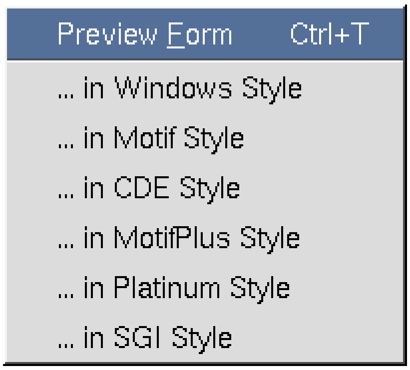


그림 1-76. Preview차림표

Preview차림표는 Alt+P에 의해 호출되며 다음과 같은 차림표선택을 제공한다.

Preview|Preview Form을 선택하여 (또는 Ctrl+T를 눌러서) Qt Designer에서 폼을 미리보기한다.

Preview|...in Windows Style을 선택하여 Windows형식으로 폼을 미리보기한다.

Preview|...in Motif Style을 선택하여 Motif형식으로 폼을 미리보기한다.

Preview|...in CDE Style을 선택하여 CDE형식으로 폼을 미리보기한다.

Preview|...in MotifPlus Style을 선택하여 MotifPlus형식으로 폼을 미리보기한다.

Preview|...in Platinum Style을 선택하여 Platinum형식으로 폼을 미리보기한다.

Preview|...in SGI Style을 선택하여 SGI형식으로 폼을 미리보기한다.

8. Window차림표

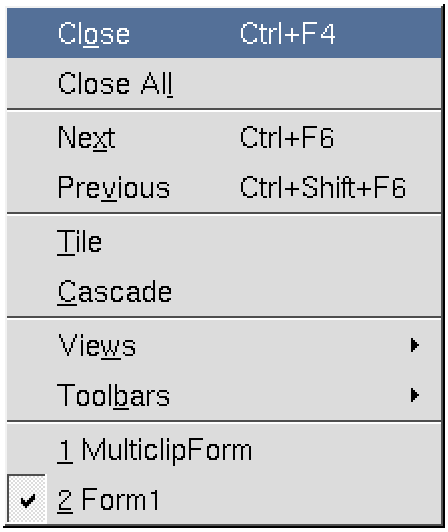


그림 1-77. Window차림표

Window차림표는 Alt+W에 의해 호출되며 다음과 같은 차림표선택을 제공한다.

Window|Close를 선택하여 (또는 Ctrl+F4를 눌러서) 현재 작업중인 창문을 닫는다.

Window|Close All을 선택하여 현재 열려져있는 창문들을 모두 닫는다.

Window|Next를 찰칵하여 (또는 Ctrl+F6을 눌러서) 다음 창문을 작업창문으로 만든다. 순서는 창문이 열려진 순서이다.

Window|Previous를 선택하여 (또는 Ctrl+Shift+F6을 눌러서) 이전 창문을 작업창문으로 만든다. 그 순서는 창문이 열려진 순서이다.

Window|Tile을 선택하여 열려진 모든 파일들과 폼들을 나란히 배치하여 매개 창문을 볼수 있게 한다.

Window|Cascade를 선택하여 열려진 모든 파일과 폼들을 각 창문의 제목띠가 보이도록 서로 겹쌓아서 탄창에 넣는다.

Window|Views|Project Overview를 선택하여 Project Overview창문을 보이게 하거나 이미 보인다면 그것을 숨긴다. 창문이 현재 보이면 검사표식이 그 차림표의 이름열에 나타난다.

Window|Views|Property Editor/Signal Handlers를 선택하여 Property

Editor/Signal Handlers창문을 보이게 하거나 이미 보이면 숨긴다. 창문이 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Views|Object Explorer를 선택하여 Object Explorer창문이 보이게 하거나 이미 보이면 숨긴다. 창문이 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Views|Line Up를 선택하여 도구띠들사이의 여유공간을 삭제하고 도구띠들이 서로 옆에 오도록 배치한다.

Window|Toolbars|File을 선택하여 File도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Toolbars|Edit를 선택하여 Edit도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Toolbars|Search를 선택하여 Search도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Toolbars|Layout를 선택하여 Layout도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Toolbars|Tools를 선택하여 Tools도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Toolbars|Buttons를 선택하여 Buttons도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Toolbars|Containers를 선택하여 Containers도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Toolbars|Views를 선택하여 Views도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Toolbars|Database를 선택하여 Database도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름옆에 나타난다.

Window|Toolbars|Input를 선택하여 Input도구띠단추들이 보이게 하거나 그것

들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름열에 나타난다.

Window|Toolbars|Display를 선택하여 Display도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름열에 나타난다.

Window|Toolbars|Custom을 선택하여 Custom도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름열에 나타난다.

Window|Toolbars|Help를 선택하여 Help도구띠단추들이 보이게 하거나 그것들이 이미 보이면 숨긴다. 도구띠단추가 현재 보이면 검사표식이 그 차림표의 이름열에 나타난다.

Window|Toolbars|Line Up를 선택하여도구띠들사이의 여유공간을 삭제하고 도구띠들이 서로 옆에 오도록 배치한다.

Window|n – 현재 열린 파일들과 폼들에 번호가 지정되여 표시된 차림표항목들중 하나를 선택하고 그것을 펼친다.

9. Help차림표

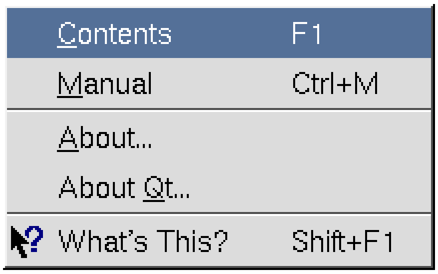


그림 1-78. Help차림표

Help차림표는 Alt+H에 의해 호출되며 다음과 같은 차림표선택을 제공한다.

Help|Contents를 선택하여 (또는 F1을 눌러서) 직결방조를 제공하는 Qt Assistant 응용프로그람을 펼친다. 직결방조는 문맥에 의존하므로 자세한 정보를 요구하는 항목을 행편집칸에 입력하면 Qt Assistant는 그것이 유효인 경우 자동적으로 탐색한다.

Help|Manual을 선택하여 (또는 Ctrl+M을 눌러서) Qt Assistant응용프로그람을 열고 방조문서를 표시한다.

Help|About를 선택하여 판번호와 허가정보를 주는 About Qt Designer대화칸을 펼친다.

Help|About Qt를 선택하여 Qt에 대한 정보를 제공하는 대화칸을 펼친다.

Help|What's This?를 선택하여 마우스지적자와 련결된 작은 물음표를 표시한다. 정보를 더 알고싶은 기능우에서 마우스를 누르면 그 기능에 대한 정보를 가진 튀여나오기차림표가 나타난다.

제11절. 도구띠단추

Qt Designer의 도구띠단추는 공통기능에 대한 고속호출을 제공한다. 도구띠단추들은 여러개의 도구띠들로 그룹화된다. 도구띠들은 왼쪽에 도구띠를 최소화하기 위하여 선택할수 있는 손잡이(handle)들을 가지고있다. 최소화된 도구띠들은 바로 차림표띠아래에 나타나는 손잡이들을 가지고있다. 이 손잡이를 선택하여 도구띠가 차지하였던 마지막위치로 되돌아갈수 있다. 도구띠의 손잡이를 끌기하여 도구띠를 도구띠령역안의 다른 위치로 옮길수 있다. 도구띠들은 도구띠령역의 밖으로 완전히 끌고나가서 독립적인 도구류동창문(tool dock window)으로 만들수 있다. 도구류동창문을 숨기려면 그 닫기단추를 찰칵한다. 숨은 도구류동창문을 되살리려면 도구령역을 오른쪽 단추로 선택하고 되살리려는 도구류동창문의 이름을 선택한다.

1. File도구띠단추

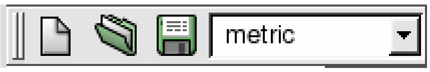


그림 1-79. File도구단추

New를 찰칵하여(또는 Ctrl+N를 눌러서) 새 프로젝트, 폼 혹은 파일을 창조한다. 이 선택은 New File대화칸을 펼친다.

Open을 찰칵하여(또는 Ctrl+O를 눌러서) 현존 프로젝트, 폼 혹은 파일을 연다. 이 단추는 파일들을 선택하는데 사용되는 File Open대화칸을 펼친다.

Save를 찰칵하여(또는 Ctrl+S를 눌러서) 프로젝트, 폼 혹은 파일들을 보관한다. 폼이나 파일들이 없는 새 프로젝트에 대하여 Save를 선택하여 완료하기전에 프로젝트를 보관한다. 새 폼들에 대하여 Save를 선택하면 Save Form As대화칸이 나타난다.

Active Project를 선택하여 현재 열려져있는 프로젝트의 이름들을 표시하고 프로젝트이름을 하나 선택하여 프로젝트들사이를 절환한다.

2. Edit도구띠단추



그림 1-80. Edit도구단추

Undo를 찰칵하여(또는 Ctrl+Z를 눌러서) 하나의 조작을 취소한다. 수행된 마지막 조작의 이름이 이 도구띠단추의 도구암시안의 Undo단어뒤에 나타난다.

Redo를 찰칵하여(또는 Ctrl+Y를 눌러서) 하나의 조작을 재시행한다. 수행된 마지막 조작의 이름이 이 도구띠단추의 도구암시의 Redo단어뒤에 나타난다.

Cut를 찰칵하여(또는 Ctrl+X를 눌러서) 현재 폼이나 파일로부터 선택항목을 삭제하고 그것을 오려둠판에 복사한다.

Copy를 찰칵하여(또는 Ctrl+C를 눌러서) 현재 폼이나 파일로부터 선택항목을 오려둠판에 복사한다.

Paste를 찰칵하여(또는 Ctrl+V를 눌러서) 오려둠판으로부터 현재 폼이나 파일에로 선택항목을(있으면) 붙이기한다.

3. Search도구띠단추



그림 1-81. Search도구단추

Find를 찰칵하여(또는 Ctrl+F를 눌러서) Find Text 대화칸을 펼친다.

Find Incremental을 찰칵하여(또는 Alt+I를 눌러서) Find도구띠단추열에 배치된 본문칸에 유표를 배치한다. 본문칸에 문자를 입력할 때마다 Qt Designer는 파일에서 처음으로 출현하는 본문을 강조표시한다. Enter건을 눌러서 다음 본문으로 이행한다. 찾으려는 단어를 발견하였으면 Esc건을 눌러서 편집기에 유표를 배치한다.

4. Tools도구띠단추

Tools도구띠단추들은 도구띠들(Qt 3.1이전)과 새로운 Toolbox에서 사용할수 있다. Qt 3.1에서 tools 도구띠들은 은페되고 Toolbox가 표시된다. Toolbox는 그룹으로 분할되는데 그룹은 제일 자주 사용하는 창문부품들을 포함하는 Common Widgets이다. 원래의 도구띠설정을 더 좋아한다면 Windows|Toolbars를 선택하고 문맥차림표에서 표시하려는 도구띠들을 선택하여 도구띠들을 호출할수 있다.

Toolbox는 도구단추들의 형을 서술하는 범주

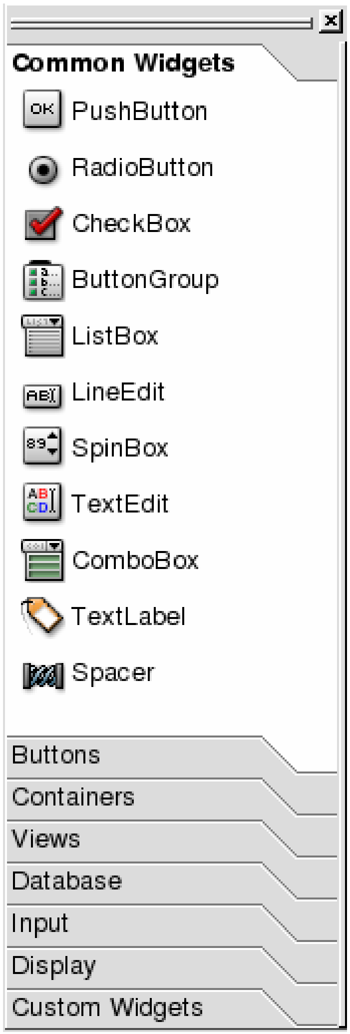


그림 1-82. Toolbox

(category)를 포함한다. 범주이름을 선택하여 그 범주안의 도구단추들을 호출할수 있다. 범주안의 창문부품을 리용하려면 창문부품을 선택하여 그것을 동작하게한다. Toolbox의 오른쪽 구석안의 X를 선택하여 닫을수 있다. Toolbox를 다시 표시하려면 Window|Views|Toolbox를 선택한다.

폼에 같은 종류의 창문부품 실례로 여러개의 누름단추를 여러개 추가하려면 그 창문부품의 도구띠단추를 두번 련속 찰칵한다. 그다음 폼을 누를 때마다 새 창문부품이 추가된다. Pointer도구띠단추를 찰칵하여 이 방식을 완료한다.

1) Tools



그림 1-83. Tools

Pointer를 찰칵하여(또는 F2을 눌러서) 선택된 위자드도구띠단추의 선택을 해제한다. Pointer는 또한 창문부품도구띠단추를 두번 련속 눌러서 새 창문부품들의 삽입을 중지하는데 리용된다. Esc건을 누르면 임의의 시각에 지적자를 복귀할수 있다.

Connect Signals and Slots를 찰칵하여(또는 F3을 눌러서) 신호와 처리부를 련결한다. 그때 창문부품을 선택하고 련결선을 련결하려는 창문부품(혹은 폼)까지 끌고간다. 마우스단추를 놓으면 View and Edit Connections대화칸이 나타난다.

Tab Order를 찰칵하여(또는 F4를 눌러서) 건반초점을 받아들일수 있는 모든 창문부품들에 대하여 타브순서를 설정한다. 이 도구띠단추를 찰칵하고 창문부품들의 곁에 나타나는 수값이 있는 푸른색원을 선택한다. 타브순서의 첫 창문부품을 선택하고 그다음 타브순서의 다음 창문부품을 선택하는 조작을 모든 창문부품들이 요구하는 타브순서번호를 가질 때까지 반복한다. 오유를 범하였으면 첫 창문부품을 두번 련속 누르고 다시 시작한다. 타브순서방식을 끝내려면 Esc를 누른다. 변경을 취소하려면 타브순서방식을 끝내고 취소한다.

Set Buddy를 찰칵하여 (또는 F12을 눌러서) 짝패를 표식자로 설정한다. 그다음 표식자를 선택하고 짝패로 하려는 창문부품까지 직선을 끌고간다. 마우스단추를 놓으면 짝패가 설정된다.

2) Buttons

PushButton을 찰칵하고 폼을 선택하여 폼에 Pushbutton을 배치한다.

ToolButton을 찰칵하고 폼을 선택하여 폼에 Toolbutton을 배치한다.

RadioButton을 선택하고 폼을 선택하여 폼우에 RadioButton을 배치한다. ButtonGroup안에 RadioButton들을 배치하여 Qt가 그룹안의 오직 하나의

RadioButton이 한번에 동작상태로 될수 있다는것을 자동적으로 담보한다.

CheckBox를 찰칵하고 폼을 선택하여 폼에 CheckBox를 배치한다.

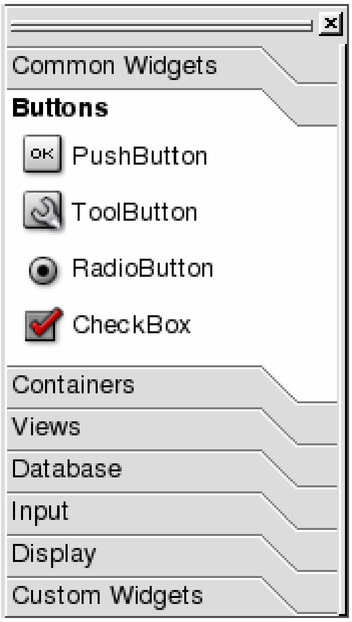


그림 1-84. Buttons

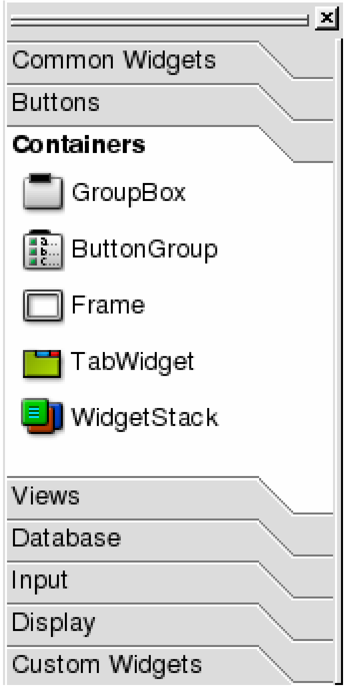


그림 1-85. Containers

3) Containers

GroupBox를 찰칵하고 폼을 선택하여 폼에 GroupBox를 배치한다.(그림 1-85)

ButtonGroup를 찰칵하고 폼을 선택하여 폼에 ButtonGroup을 배치한다.

Frame을 찰칵하고 폼을 선택하여 폼에 Frame을 배치한다.

TabWidget를 선택하고 폼을 선택하여 폼우에 TabWidget를 배치한다. 타브를 추가 혹은 삭제하려면 타브창문부품을 오른쪽 단추로 찰칵하고 Add Page 혹은 Remove Page를 선택한다.

4) Views

ListBox를 찰칵하고 폼을 선택하여 폼에 ListBox를 배치한다.(그림 1-86)

ListView를 찰칵하고 폼을 선택하여 폼에 ListView를 배치한다.

Icon View를 찰칵하고 폼을 선택하여 폼에 IconView를 배치한다.

Table을 찰칵하고 폼을 선택하여 폼에 Table을 배치한다.



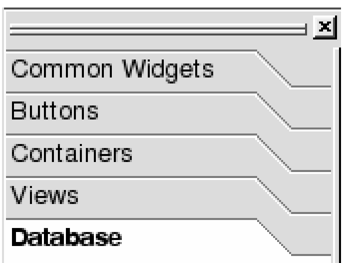


그림 1-87. Database

5) Database

DataTable을 찰칵하고 폼을 선택하여 폼에 DataTable을 배치한다.(그림 1-87)

DataBrowser를 찰칵하고 폼을 선택하여 폼에 DataBrowser을 배치한다.

DataView를 찰칵하고 폼을 선택하여 폼에 DataView을 배치한다.

6) Input

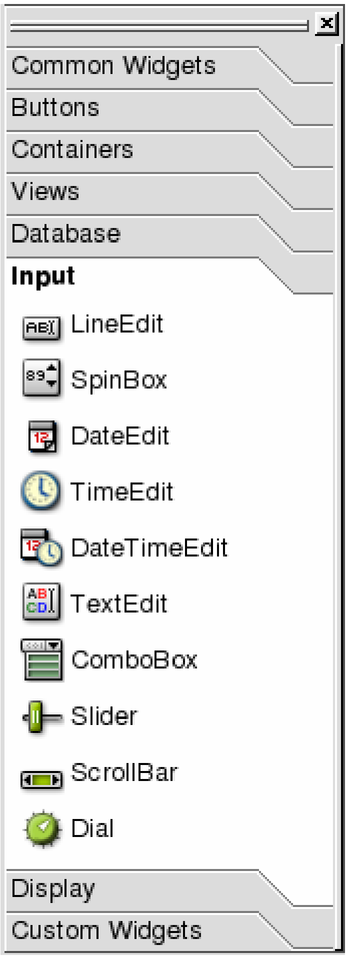


그림 1-88. Input 도구단추

LineEdit를 찰칵하고 폼을 선택하여 폼에 LineEdit를 배치한다.
SpinBox를 찰칵하고 폼을 선택하여 폼에 SpinBox를 배치한다.
DateEdit를 찰칵하고 폼을 선택하여 폼에 DateEdit를 배치한다.
TimeEdit를 찰칵하고 폼을 선택하여 폼에 TimeEdit를 배치한다.
DateTimeEdit를 찰칵하고 폼을 선택하여 폼에 DateTimeEdit를 배치한다.
TextEdit를 찰칵하고 폼을 선택하여 폼에 TextEdit를 배치한다.
ComboBox를 찰칵하고 폼을 선택하여 폼에 ComboBox를 배치한다.
Slider를 찰칵하고 폼을 선택하여 폼에 Slider를 배치한다.
ScrollBar를 찰칵하고 폼을 선택하여 폼에 Scrollbar를 배치한다.
Dial을 찰칵하고 폼을 선택하여 폼에 Dial을 배치한다.

7) Display

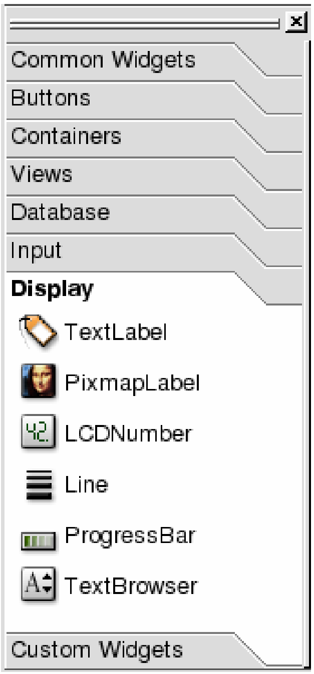


그림 1-89. Display 도구단추

TextLabel을 찰칵하고 폼을 선택하여 폼에 TextLabel을 배치한다.

PixmapLabel을 찰칵하고 폼을 선택하여 폼에 PixmapLabel을 배치한다.

LCDNumber를 찰칵하고 폼을 선택하여 폼에 LCDNumber를 배치한다.

Line을 찰칵하고 폼을 선택하여 폼에 Line을 배치한다.

ProgressBar를 찰칵하고 폼을 선택하여 폼에 ProgressBar를 배치한다.

TextBrowser를 찰칵하고 폼을 선택하여 폼에 TextBrowser을 배치한다.

8) Custom

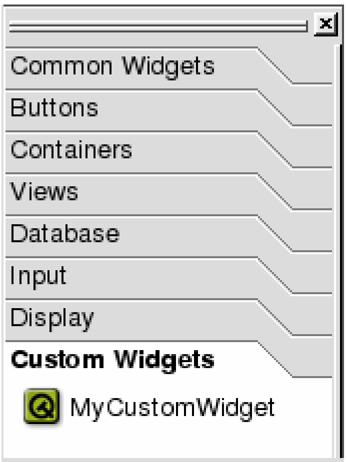


그림 1-90. Custom Widget도구단추

My Custom Widget를 찰칵하고 폼을 선택하여 폼에 Custom Widget를 배치한다. 이 도구띠단추는 Tools|Custom|Edit Custom Widgets를 사용하여 사용자정의창문부품을 창조한 경우에만 나타난다.

9) Layout 도구띠단추

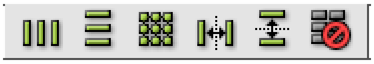


그림 1-91. Layout도구단추

Adjust Size를 선택하여 (또는 Ctrl+J를 눌러서) 요구하는 크기로 창문부품의 크기를 조절한다.

Lay Out Horizontally를 선택하여 (또는 Ctrl+H를 눌러서) 선택된 창문부품 혹은 배치들을 배치한다. Shift를 누르고 찰칵하여 각 창문부품 혹은 배치를 선택하고 이 차림표선택을 지정하여 그것들을 수평으로 묶는다. 흔히 복잡한 창문부품들에 대하여 Object Explorer창문의 Widgets타브에서 창문부품과 배치들을 눌러서 그것들을 선택하는것이 가장 간단하다. 오직 한개의 창문부품이 선택되면 그 자식창문부품들이 수평으로 배치된다.

Lay Out Vertically를 선택하여(또는 Ctrl+L을 눌러서) 선택된 창문부품들을 배치한다. Shift를 누르고 찰칵하여 매개 창문부품 혹은 배치를 선택한 다음 이 차림표항목을 선택하여 그것들을 수직으로 묶는다. 흔히 복잡한 창문부품들에 대하여 Object Explorer창문의 Widgets타브에서 창문부품과 배치들을 눌러서 그것들을 선택하는것이 가장 간단하다. 오직 한개의 창문부품이 선택되면 그 자식창문부품들이 수직으로 배치된다.

Lay out in a Grid를 선택하여(또는 Ctrl+G를 눌러서) 선택된 창문부품들을 살창으로 배치한다. 오직 한개의 창문부품이 선택되면 그 자식창문부품들이 살창으로 배치된다.

Lay Out Horizontally (in Splitter)를 선택하여 선택된 창문부품이나 배치들의 매개사이에 분할기(splitter)를 주어서 배치한다. Shift를 누르고 찰칵하여 각 창문부품이나 배치를 선택하고 이 차림표항목을 선택하여 그것들을 수평으로 묶는다. 흔히 복잡한 창문부품들에 대하여 Object Explorer창문의 Widgets타브에서 창문부품과 배치들을 눌러서 그것들을 선택하는것이 가장 간단하다.

Lay Out Vertically (in Splitter)를 선택하여 선택된 창문부품이나 배치들의 매개사이에 분할기를 주어서 배치한다. Shift를 누르고 찰칵하여 각 창문부품이나 배치를 선택하고 이 차림표항목을 선택하여 그것들을 수직으로 묶는다. 흔히 복잡한 창문부품들에 대하여 Object Explorer창문의 Widgets타브에서 창문부품과 배치들을 눌러서 그것들을 선택하는것이 가장 간단하다.

Break Layout를 선택하여(또는 Ctrl+B를 눌러서) 배치를 파괴한다. 배치를 누르

고 이 항목을 선택하면 배치가 삭제된다.

Add Spacer를 선택하여 폼에 너무 많은 공간을 가지는 창문부품들에 수직 혹은 수평수축자를 추가한다. 수축자는 배치에서 여유공간을 차지한다.

5. Help도구띠단추

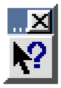


그림 1-92. Help도구단추

What's This?를 선택하여 마우스지적자와 련결된 작은 물음표를 표시한다. 정보를 더 알고싶은 기능우에서 마우스를 누르면 그 기능에 대한 정보를 가진 튀여나오기차림표가 나타난다.

제12절. 대화칸

이 절에서는 Qt Designer의 매개 대화칸에 대하여 설명한다.

1. Menu대화칸

1) Qt Designer New/Open대화칸

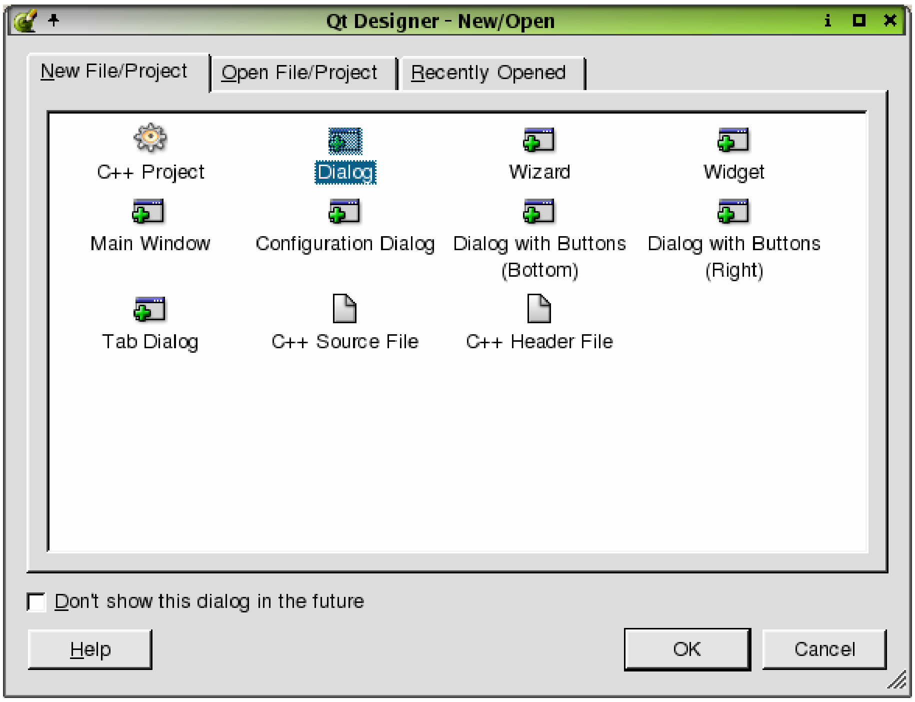


그림 1-93. Qt Designer New/Open

Qt Designer를 기동하면 Qt Designer New/Open대화칸이 열린다. 이 대화칸은

Qt Designer를 기동하여 무엇을 하려고 하는가에 따라서 사용하기 위한 3개의 타브를 포함한다.

- New File/Project타브

이 타브는 Qt Designer를 기동할 때 기정으로 되여있다. 새 프로젝트를 기동하거나 혹은 창문으로부터 적당한 그림기호를 선택하여 파일을 창조할 때 선택한다. 사용할 수 있는 각이한 파일형에 대한 자세한 정보는 다음 항을 참고하시오.

다음번에 이 대화칸을 숨기려면 "Don't show this dialog in the future"검사칸을 설정하고 Qt Designer를 기동한다.

- Open File/Project타브

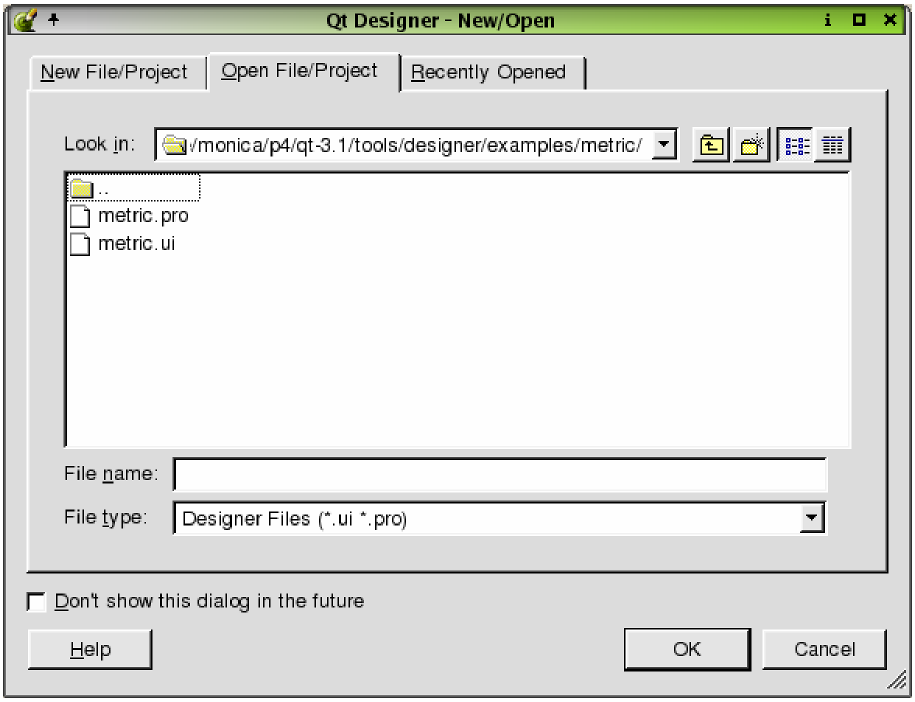


그림 1-94. Open File/Project

이 타브를 선택하여 이미 존재하는 파일이나 프로젝트를 꺼낸다. 타브는 현재 등록부와 기정파일형을 보여준다. 다른 등록부를 선택하려면 Look In결합칸을 선택한다. 파일을 선택하면 그 이름이 File Name복합칸에 나타난다. 다른 파일형을 선택하려면 File Type복합칸을 선택한다. Create New Folder도구띠단추를 찰칵하여 새 등록부를 창조한다. List View도구띠단추를 찰칵하여 이름만 보여주는 목록에 홀더와 파일들을 표시한다. Details도구띠단추를 찰칵하여 폴더와 파일들을 그 이름, 형, 날자, 속성들과 함께 표시한다. Size, Type, Date, 또는 Attributes렬제목을 선택하여 폴더 혹은 파일들을 정렬한다.

OK를 찰칵하여 선택된 파일을 연다. Cancel을 찰칵하여 새 파일을 열지 않고 대화

칸을 닫는다.

- Recently Opened타브

이 타브는 창문에 최근에 연 파일들이나 프로젝트들을 모두 현시한다. 파일이나 프로젝트를 선택하고 OK를 찰각하여 연다.

2. File대화칸

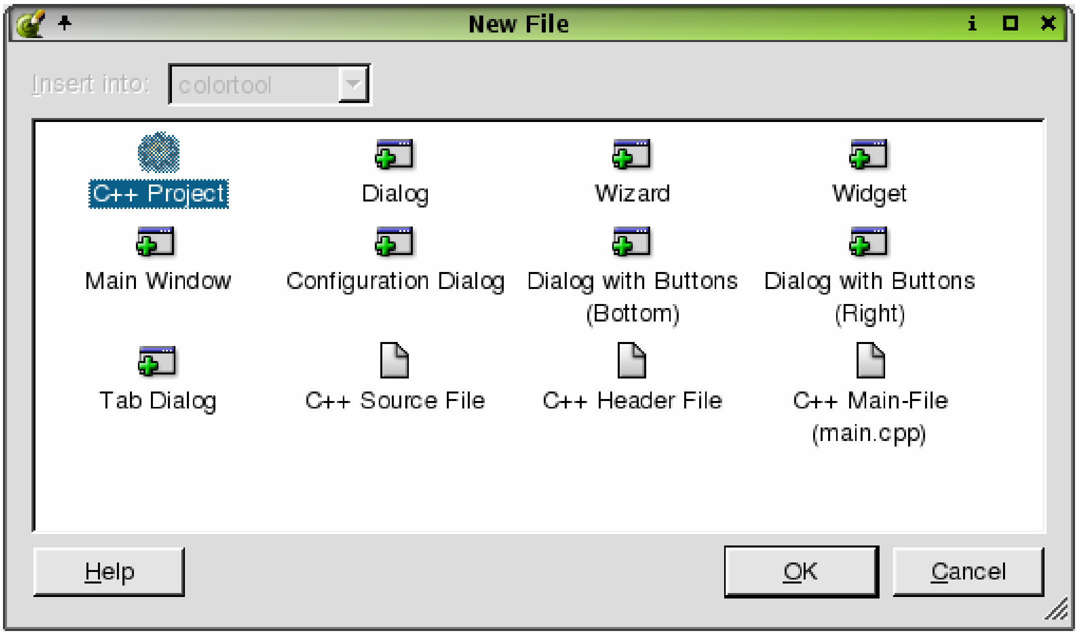


그림 1-95. New File

1) New File대화칸

File|New를 선택하여(혹은 Ctrl+N을 눌러서) New File대화칸을 펼친다. 이 대화칸은 선택할수 있는 4종의 파일들을 제공한다. 즉 C++프로젝트, 폼, 원천파일, 그리고 기본파일.

Insert Into내리펼침복합칸은 현재 프로젝트에 기정으로 되여있는 열려진 프로젝트들을 렬거한다. 이 복합칸에 표시된 프로젝트에 새 파일들을 추가한다. 각이한 프로젝트에 새 파일을 추가하려면 Insert Into복합칸에서 사용하려는 프로젝트를 선택한다.

Dialog파일형은 New File대화칸이 펼쳐질 때 기정으로 선택된다. 사용하려는 파일형을 선택하고 OK를 찰각하여 창조한다. Cancel을 찰각하여 새 파일을 창조하지 않고 대화칸을 닫을수 있다. C++프로젝트를 선택하면 Insert Into복합칸은 허용되지 않으므로 현존프로젝트에 새로운 C++프로젝트를 삽입할수 없다.

C++ Project를 선택하여 새 프로젝트를 시작한다. 이 선택은 Project Settings대화칸을 펼친다. C++프로젝트들은 .pro파일로서 보관되며 여기에는 Qt Designer가 프로젝트들을 관리하기 위하여 요구하는 정보들이 포함된다. Qt Designer에서 자기 프로젝트

에 폼을 추가할 때 그것은 자동적으로 프로젝트파일의 FORMS절에 추가된다. .pro파일은 프로젝트에서 사용된 폼들(.ui파일들)의 목록을 포함한다. Qt Designer는 .ui파일(실례로 form.ui)들을 읽고쓴다. uic(사용자대면부콤파일러)는 .ui파일로부터 머리부파일(실례로 form.h)과 실현파일(실례로 form.cpp)을 둘다 창조한다.

Dialog을 선택하여 일반대화칸폼을 창조한다. 전형적으로 이 형의 폼은 사용자에게 환경선택을 제시하거나 관련된 선택모임(실례로 인쇄기설정대화칸와 탐색 및 교체대화칸)을 제시하는데 리용된다.

Wizard를 선택하여 위자드폼을 창조한다. 위자드는 일련의 대화칸페지들로 이루어지는 특수한 형태의 입력대화칸이다. 위자드의 목적은 사용자가 어떤 과정을 한걸음씩 추적하여 일감을 자동화하도록 사용자를 방조하는것이다. 위자드는 사람들이 배우기 힘들거나 수행하기 힘든 복잡한 과제나 드물게 제기되는 과제들에 사용할수 있다. 처음에 위자드폼은 하나의 대화칸페지로 구성된다. 오른쪽 단추를 리용하여 문맥차림표를 선택하고 추가적인 페지들을 첨부하고 페지제목들을 변경한다.

Widget를 선택하여 기초클라스가 QDialog가 아니라 QWidget인 폼을 창조한다.

Main Window를 선택하여 Main Window Wizard를 펼친다. 이 위자드는 작용과 차림표선택, 사용자가 작용을 호출할수 있는 도구띠를 창조하는데 리용된다. 이 폼은 전형적인 기본창문형식 응용프로그람을 창조하는데 사용된다.

Configuration Dialog를 선택하여 왼쪽에 목록칸, 폼을 꽉 채우는 타브창문부품 그리고 Help와 OK, Cancel단추들을 가지는 폼을 창조한다.

Dialog with Buttons (Bottom)폼은 폼의 바닥에 기정단추들이 있는 형판이다.

Dialog with Buttons (Right)폼은 폼의 오른쪽에 기정단추들이 있는 형판이다.

Tab Dialog폼은 중심창문부품으로서 타브창문부품과 함께 바닥에 Help, OK, Cancel단추들이 있다.

C++ Source File을 선택하여 빈 C++파일을 새로 창조한다. 이 파일은 그것을 보관할 때 자동적으로 프로젝트에 추가된다.

C++ Header File을 선택하여 빈 C++머리부파일을 새로 창조한다. 이 파일은 그것을 보관할 때 자동적으로 프로젝트에 추가된다..

C++ Main File을 선택하여 기본main.cpp파일을 자동적으로 창조하는 Configure Main-File대화칸을 펼친다. C++기본파일은 열려있는 프로젝트가 없으면 가능한 선택이 아니다. 또한 이 파일형은 프로젝트를 창조하지 않고서는 main.cpp파일을 창조할수 없으므로 Qt Designer New/Open대화칸에서는 사용할수 없다.

2) Configure Main-File대화칸

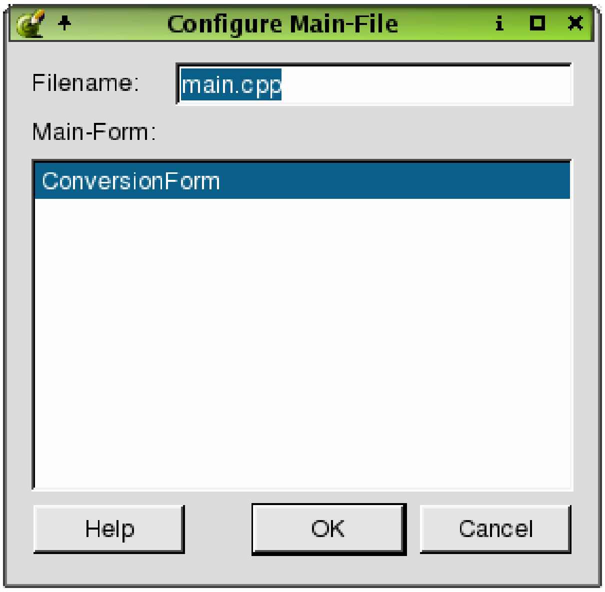


그림 1-96. Configure Main-File

File|New|C++ Main-File을 선택하여 Configure Main-File대화칸을 펼친다. 이 대화칸을 리용하여 기본파일과 그 폼들의 환경을 구성한다.

기정파일이름을 변경하려면 Filename행편집칸에 그것을 입력한다. 폼을 선택하고 행편집칸으로부터 그것을 선택함으로써 응용프로그람의 기본폼으로 사용한다.

OK를 찰칵하여 환경구성을 받아들이면 Qt Designer가 기정의 main.cpp 파일을 창조한다. Cancel을 찰칵하여 대화칸을 닫는다.

자료기지프로그람작성자들을 위한 알아두기: Qt Designer로 main.cpp파일을 창조하면 이 파일은 createConnections()함수를 포함하지 않는다. 자료기지련결에는 사용자이름과 통과암호가 필요하기때문에 이 함수를 포함하지 않는다. 결과로서 Qt Designer에서 정확히 예견하는 응용프로그람들은 자체의 자료기지련결함수를 실현하지 않으면 실행할수 없다.

3) File Open대화칸

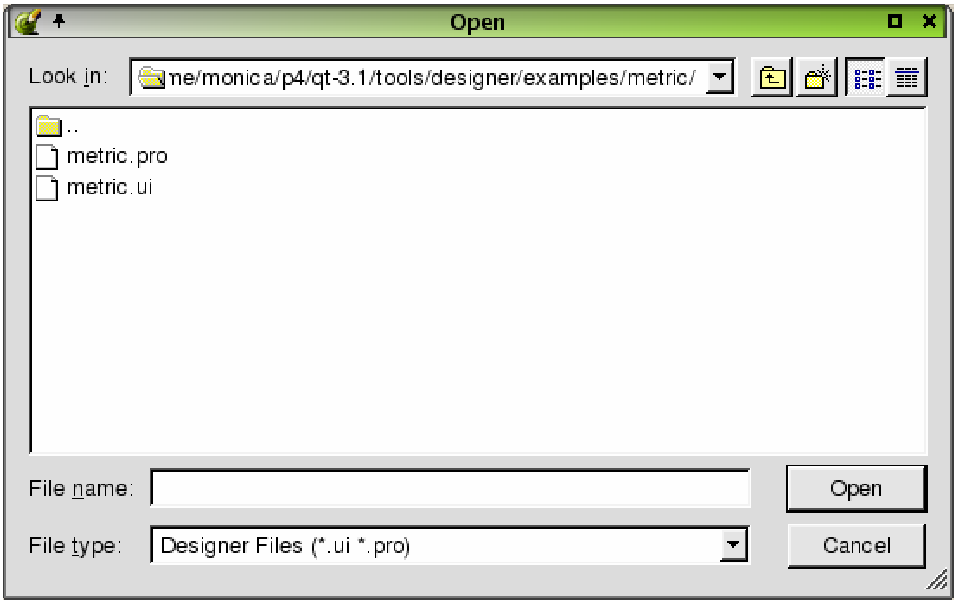


그림 1-97. File Open

File|Open을 선택하여(Ctrl+O를 눌러서) Open대화칸을 연다. 이 대화칸에 의하여 현존파일들을 연다.

Open대화칸은 현재등록부와 기정파일형을 표시한다. 다른 등록부를 선택하려면 Look In복합칸을 선택한다. 파일을 선택하면 그 이름이 File Name복합칸에 나타난다. 다른 파일형을 선택하려면 File Type복합칸을 선택한다. Create New Folder도구띠단추를 찰칵하여 새 등록부를 창조한다. List View도구띠단추를 찰칵하여 목록에 폴더와 파일들의 이름만 표시한다. Details도구띠단추를 찰칵하여 폴더와 파일이름, 그리고 그 크기, 형, 속성들을 표시한다. Size, Type, Date, 혹은 Attributes렬의 머리부를 선택하여 폴더 혹은 파일들을 정렬한다.

Open을 눌러서 선택된 파일을 연다. Cancel을 찰칵하여 새 파일을 열지 않고 대화칸을 닫는다.

알아두기: Windows에서는 System File대화칸이 사용된다.

4) Save As대화칸

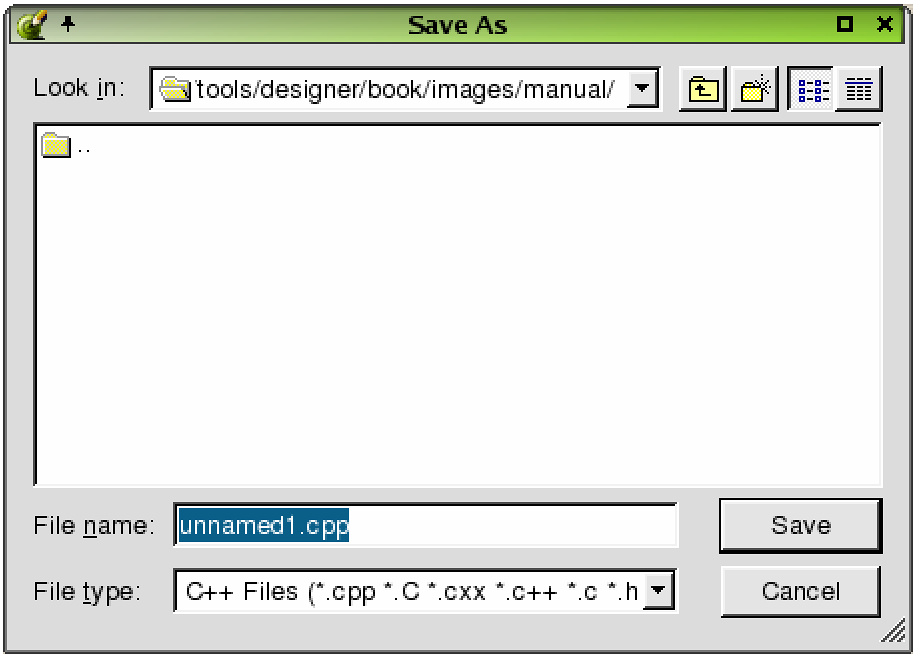


그림 1-98. Save As대화칸

File|Save As를 선택하여 Save As대화칸을 펼친다. 이 대화칸에 의하여 등록부에 파일들을 보관한다.

Save As대화칸은 현재 등록부와 기정파일형을 표시한다. 다른 등록부를 선택하려면 Look In복합칸을 찰칵한다. 파일을 선택하면 그 이름이 File Name복합칸에 나타난다. 다른 파일형을 선택하려면 File Type복합칸을 찰칵한다. Create New Folder도구띠단추를 찰칵하여 새 등록부를 창조한다. List View도구띠단추를 찰칵하여 목록에 폴더와 파일들의 이름만 표시한다. Details도구띠단추를 찰칵하여 폴더와 파일이름, 그리고 그 크기, 형, 날자, 속성들을 표시한다. Size, Type, Date, 혹은 Attributes렬제목을 선택하여 폴더나 파일들을 정렬한다.

Save를 선택하여 선택된 파일을 보관한다. Cancel을 찰칵하여 파일을 보관하지 않고 대화칸을 닫는다.

알아두기: Windows에서는 System File대화칸이 사용된다.

5) Save Form As대화칸

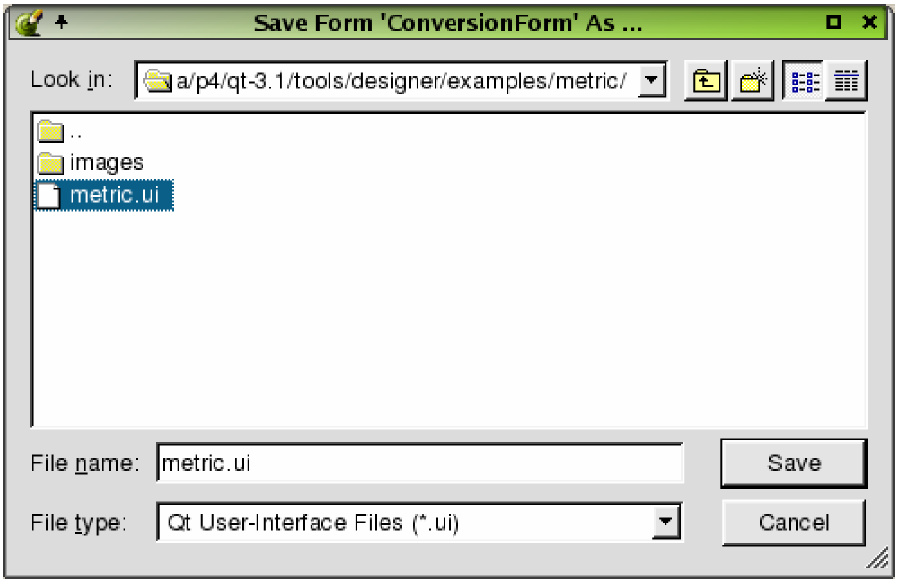


그림 1-99. Save Form As

폼을 보관할 때 File|Save As를 선택하여 Save Form As를 연다.

Save Form As대화칸은 현재등록부와 기정파일형을 표시한다. 다른 등록부를 선택하려면 Look In복합칸을 찰칵한다. 파일을 선택하면 그 이름이 File Name복합칸에 나타난다. 다른 파일형을 선택하려면 File Type복합칸을 찰칵한다. Create New Folder도구띠단추를 찰칵하여 새 등록부를 창조한다. List View도구띠단추를 찰칵하여 폴더와 파일들의 이름만 목록에 표시한다. Details도구띠단추를 찰칵하여 폴더와 파일이름, 그리고 그 크기, 형, 날자, 속성들을 표시한다. Size, Type, Date, 혹은 Attributes렬제목을 선택하여 폴더나 파일들을 정렬한다.

Save를 눌러서 선택된 폼을 보관한다. Cancel을 찰칵하여 폼을 보관하지 않고 대화칸을 닫는다.

6) Create Template대화칸

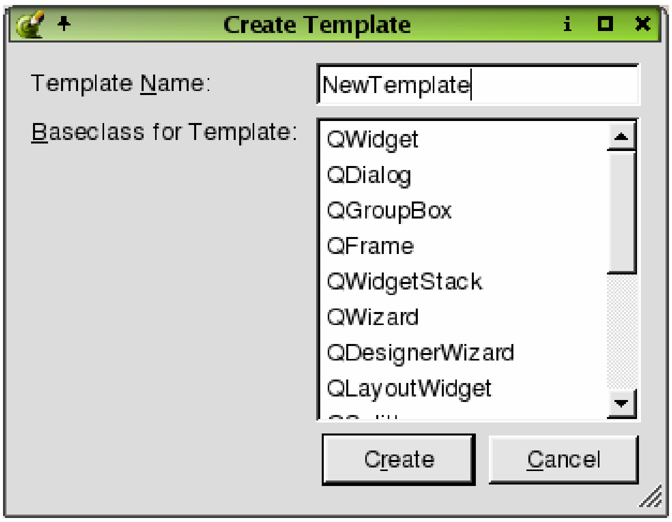


그림 1-100. Create Template대화칸

File|Create Template를 선택하여 Create Template대화칸을 펼친다. 이 대화칸을 리용하여 형판을 창조한다.

Template Name행편집칸의 기정은 New Template이다. 다른 이름으로 변경하려면 그것을 행편집칸에 입력한다. Baseclass for Template흐름띠를 누르고 형판의 기초클라스를 선택한다.

Create를 선택하여 형판을 창조한다. Cancel을 찰칵하여 형판을 창조하지 않고 대화칸을 닫는다.

형판을 창조하면 그것이 New File대화칸에 나타난다. 형판는 비슷한 폼을 대량적으로 생성해야 할 때 혹은 자기 폼들에 형타를 만들려고 할 때 사용할수 있다.

3. Edit대화칸

1) Edit Functions대화칸

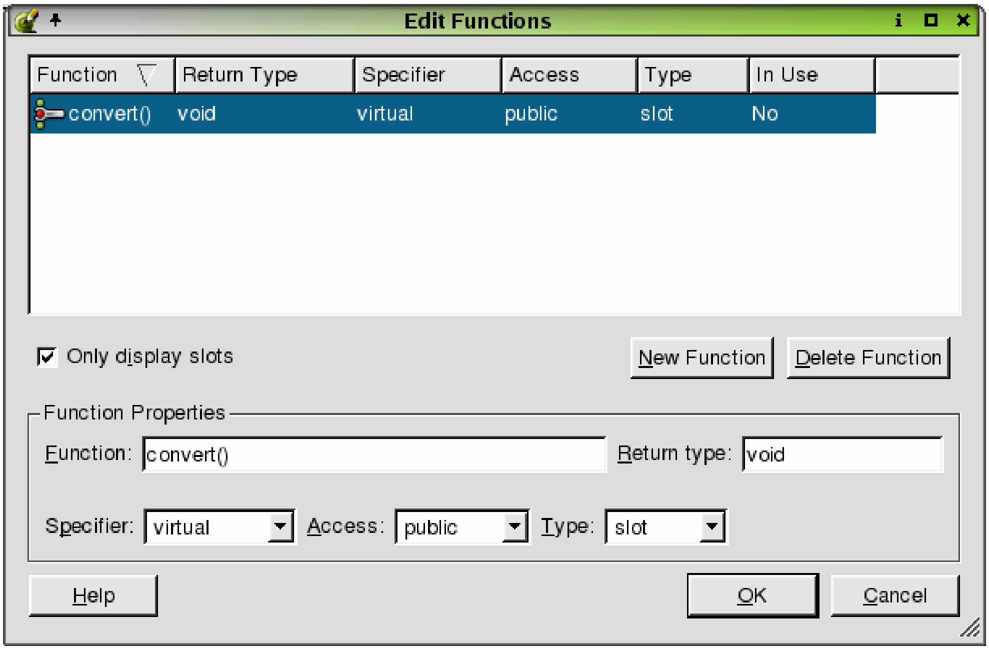


그림 1-101. Edit Functions대화칸

Edit|Slots를 선택하여 Edit Functions대화칸을 펼친다. 이 대화칸에 의하여 신호와 결합하여 객체들사이의 통신을 제공하는데 쓰이는 처리부나 함수들을 편집하거나 창조한다.

이 대화칸이 펼쳐지면 모든 현존처리부와 함수들이 목록보기에 표시된다. 렬제목 Function, Return Type, Specifier, Access, Type 그리고 In Use는 렬거되는 각 함수에 대하여 자세한 정보를 제공한다. 임의의 렬제목을 선택하여 함수들을 정렬한다. 새 함수를 창조하려면 New Function단추를 찰칵한다. 새 함수는 기정이름을 가지므로 Function행편집칸에 새 이름을 입력하여 교체하여야 한다. Return Type도 역시 기정이므로 행편집칸에 입력하여 변경할수 있다. Specifier 혹은 Access를 변경하려면 복합

칸을 누르고 필요한 지정자나 호출을 선택한다. 함수의 형(함수 혹은 처리부)을 변경하려면 Type복합칸을 선택한다. 함수를 삭제하려면 삭제하려는 함수를 선택하고 Delete Function단추를 찰칵한다.

OK를 찰칵하여 함수에 대한 변경을 모두 보관한다. Cancel을 찰칵하여 함수에 대한 변경을 보관하지 않고 대화칸을 닫는다.

2) View and Edit Connections대화칸

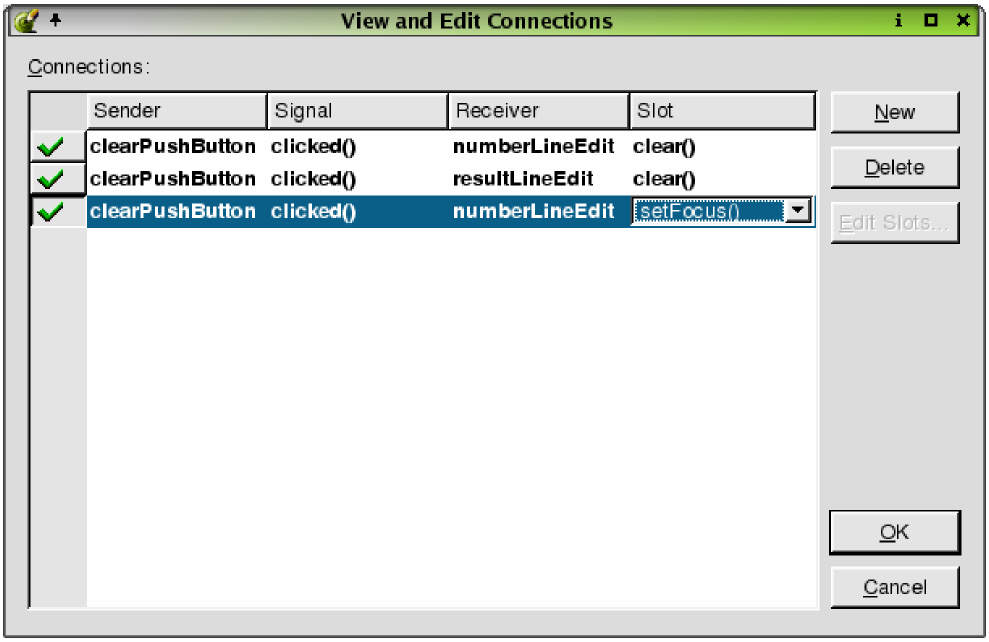


그림 1-102. View and Edit Connections대화칸

Edit|Connections를 선택하여 View and Edit Connections대화칸을 펼친다. 이 대화칸도 역시 임의의 창문부품을 오른쪽 단추로 선택하여 호출된다. 이 대화칸을 리용하여 신호-처리부련결을 표시하고 편집한다.

이 대화칸이 열리면 모든 현존련결이 목록칸에 표시된다. 렬제목 Sender, Signal, Receiver, 그리고 Slot는 각 련결에 대한 자세한 정보를 제공한다. 렬제목을 선택하여 련결을 정렬한다. 새 련결을 추가하려면 New단추를 찰칵한다. 련결의 Sender, Signal, Receiver 그리고 Slot를 지정하려면 련결의 적당한 마당들을 누르고 복합칸에서 선택한다. 련결을 삭제하려면 련결을 선택하고 Delete단추를 찰칵한다. 현재 폼을 위한 전용처리부를 편집하려면 현재 선택된 련결에서 Receiver로서 폼을 선택하고 Edit Slots 단추를 찰칵한다. 이것은 Edit Functions대화칸을 펼친다.

OK를 찰칵하여 련결에 대한 변경을 모두 보관한다. Cancel을 찰칵하여 련결에 대한 변경을 보관하지 않고 대화칸을 닫는다.

3) Form Settings대화칸

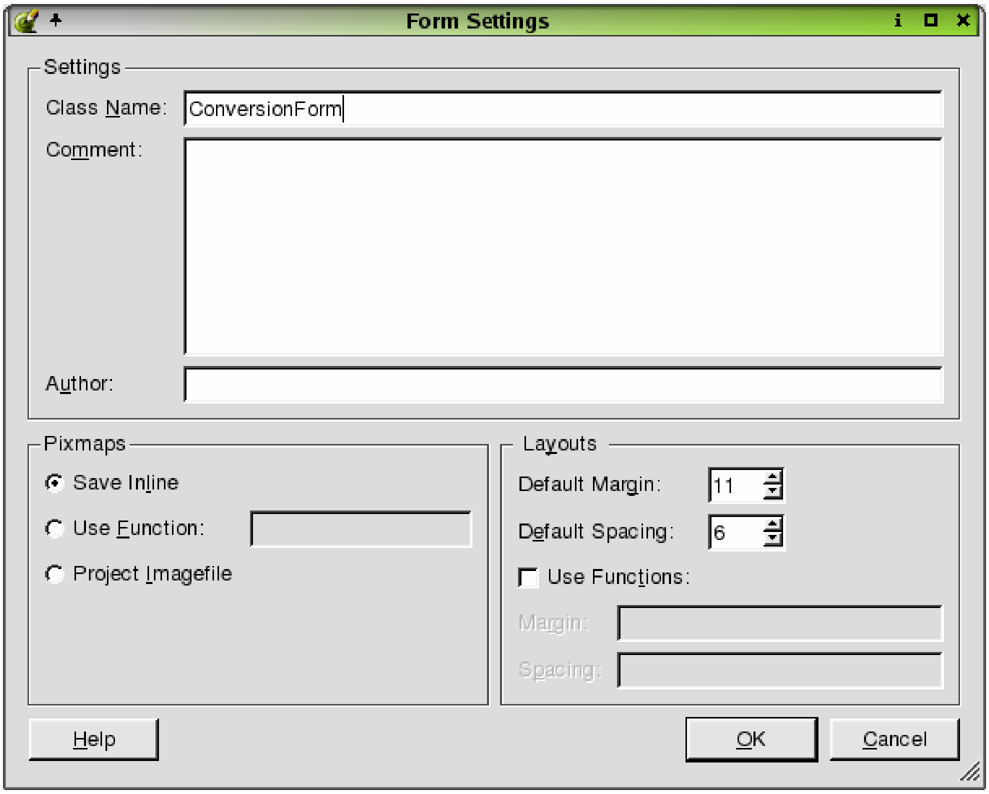


그림 1-103. Form Settings대화칸

Edit|Form Settings를 선택하여 Form Settings대화칸을 펼친다. 이 대화칸을 리용하여 폼의 설정, 픽스매프, 그리고 배치속성을 보관한다.

- Settings

Settings절에서 Class Name행편집칸에 입력하여 창조한 클라스의 이름을 변경하거나 추가할수 있다. 기정이름은 폼이름이지만 변경할수 있다. 또한 Comment와 Author행편집칸에 본문을 입력하거나 그것들이 필요없다면 비워둘수 있다.

- Pixmaps

기정(프로젝트들에서)은 Project Imagefile이다. 이것은 권고하는 선택이다. 화상은 자동적으로 조절되며 Qt Designer를 리용하여 보조등록부에서 화상들을 기억하고 uic는 화상을 포함하는 코드와 필요한 지원코드를 생성한다. 매개 화상은 한번만 기억되며 여러 폼에서 리용할수 있다.

Qt Designer로 화상들을 조종하려고 하지 않는다면 (혹은 프로젝트를 사용하고있지 않는다면) Save Inline 혹은 Use Function을 선택한다. Save Inline은 .ui 파일들에 픽스매프를 보관한다. 이 수법의 결함은 화상들이 사용되는 폼들에 그것들을 보관한다는것이며 이것은 화상들을 폼들에서 교차로 공유할수 없다는것을 의미한다. Use Function을 선택하여 픽스매프의 적재에 자체의 그림기호적재기함수를 리용한다. Use

Function행편집칸에 함수의 이름(서명이 없이)을 입력한다. 이 함수는 픽스매프적재를 위하여 생성된 코드에서 리용된다. 자기의 함수는 화상이 요구될 때마다 픽스매프속성에 입력한 본문(실례로 화상이름)과 함께 호출된다.

- Layouts

Default Margin스핀칸 혹은 Default Spacing스핀칸을 선택하여 현재 폼의 기정배치설정을 변경한다.

배치의 기정여백과 공백용의 값들을 동적으로 얻기 위하여 생성한 코드에서 함수를 리용하려면 Use Functions검사칸을 설정한다. Margin과 Spacing행편집칸들에서 여백과 공백을 얻는데 사용해야 할 함수이름(서명이나 parantheses없이)을 지정한다.

OK를 찰각하여 폼설정에 대한 변경을 받아들인다. Cancel을 찰각하여 변경을 보관하지 않고 대화칸을 닫는다.

4) Preferences대화칸

Edit|Preferences를 선택하여 Preferences 대화칸을 펼친다. 이 대화칸에는 General선택용 타브가 있다. C++ Editor플라그인을 가지고있으면 대화칸에는 C++ Editor타브도 있다.

- General타브

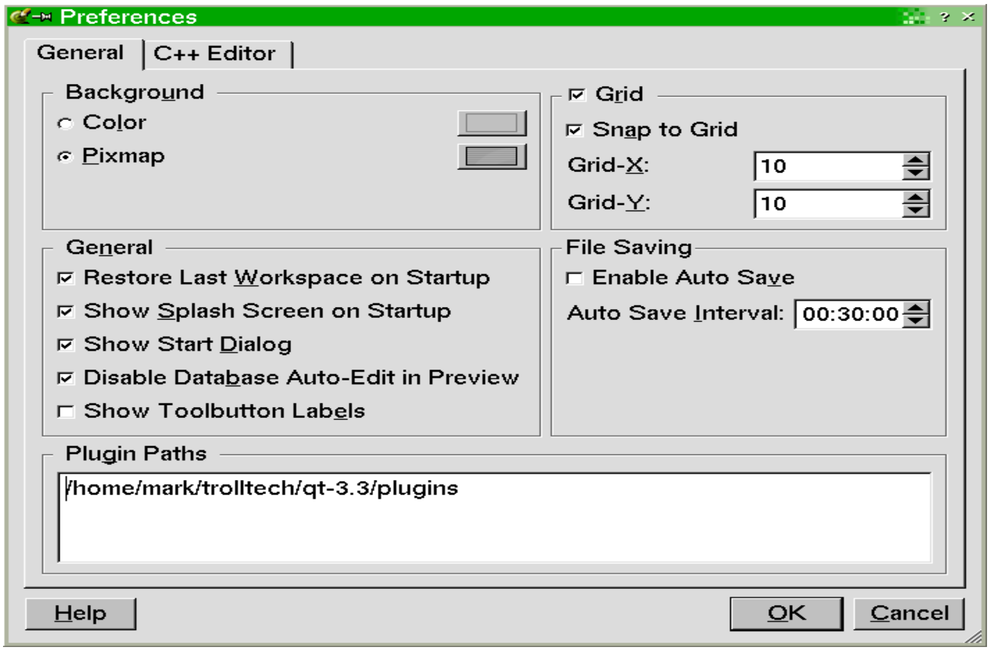


그림 1-104. Preferences- General타브

General타브에는 Background와 Grid, General, File Saving, Plugin Paths

부분들이 있다.

Background부분은 기정으로 Pixmap로 되여있다. 기정값을 변경하려면 Pixmap라지오단추옆의 Select a Pixmap단추를 찰칵하여 Choose a Pixmap...대화칸을 펼친다. Color라지오단추를 찰칵하여 배경을 픽스매프대신에 색으로 변경할수도 있다. Color라지오단추의 오른쪽에 있는 Choose a Color단추를 찰칵하여 Select Color대화칸을 펼친다.

Grid부분에는 폼에서 살창을 전용화하기 위한 선택이 있다. Grid부분에 배치된 Show Grid검사칸은 기정으로 선택된 상태이다. Qt Designer를 리용하는 개발자들은 거의 항상 Qt의 배치를 리용하여 자기의 폼들을 설계하고 드문히 살창을 리용한다. 폼이 고정크기와 위치를 가지는 창문부품들을 리용하여 창조되는 경우에 드물게 살창이 제공된다. Show Grid가 선택되여있으면 살창의 모양을 전용화할수 있다. 그것이 선택되지 않았으면 Grid부분이 허용되지 않는다. Snap to Grid검사칸도 역시 기정으로 설정되여있다. 그것이 선택되여있을 때 창문부품들은 X|Y해상도를 리용하여 점에 배치된다. 그것이 검사되지 않을 때 Grid-X와 Grid-Y스핀칸들은 허용되지 않는다. Grid X와 Grid Y스핀칸들을 선택하여 모든 폼용으로 살창설정을 전용화할수 있다.

General타브의 General부분에는 4개의 검사칸이 있다. Restore last workspace on startup검사칸을 설정하여 Qt Designer의 창문과 도구띠들의 크기와 위치를 보관한다. Qt Designer가 다음에 기동할 때 창문과 도구띠들이 그 마지막 위치에 복귀된다. Show Splash Screen on startup검사칸을 설정하여 응용프로그람을 기동할 때 Qt Designer기동화면을 표시한다. Disable Database Auto-Edit in Preview검사칸을 설정하여 자료기지창문부품들과 작업할 때 련결되는 자료기지에서 자료를 갱신하거나 삭제하는 기능을 허용하지 않는다. Show Toolbutton Label를 선택하여 Qt Designer의 도구띠단추들아래에 본문표식자가 나타나게 한다.

Qt Designer가 파일을 자동적으로 보관하게 하려면 Enable Auto Save검사칸을 설정하고 시격을 설정한다.

Qt Designer는 플라그인경로에서 발견한 플래그인들을 적재한다. 자기의 전용플라그인경로들을 추가하려면 그것들을 플라그인경로 다중행편집칸에 입력한다. (한행에 하나씩)

- C++ Editor타브

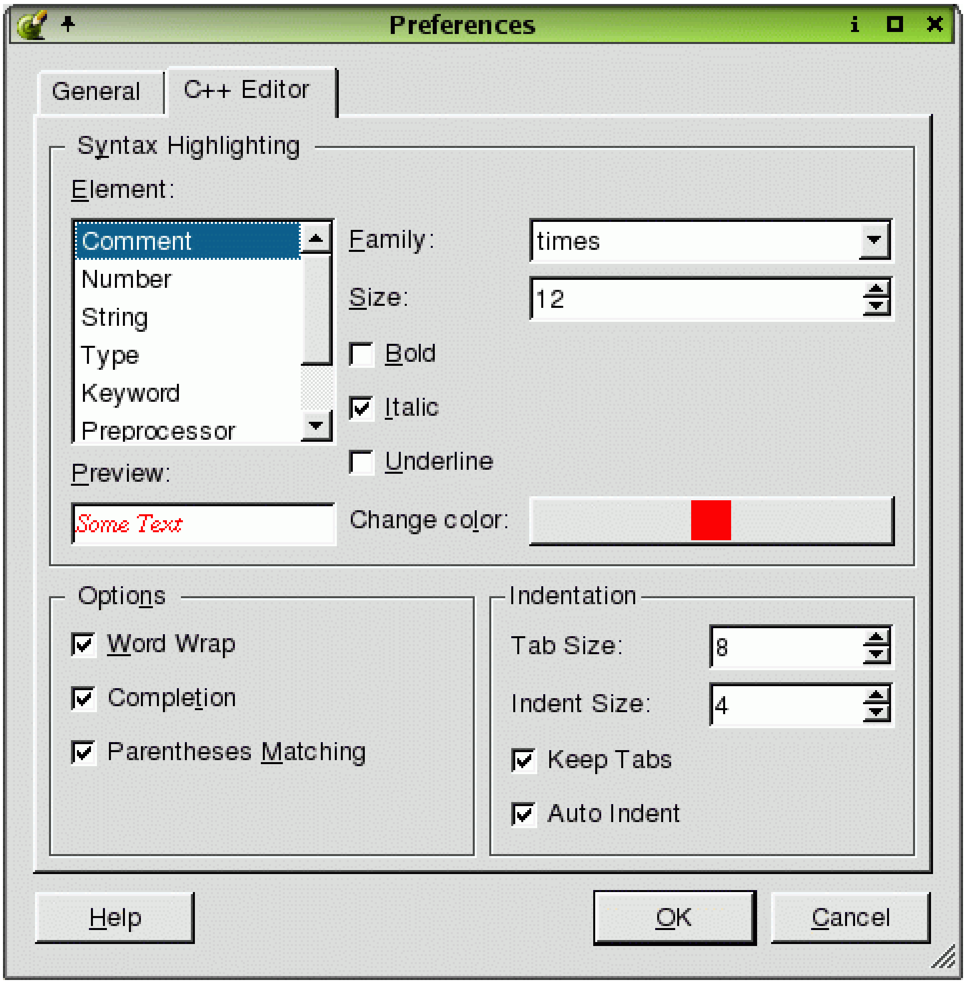


그림 1-105. Preferences- C++ Editor타브

C++ Editor타브는 편집기를 전용화하기 위한 선택을 제공한다. Syntax Highlighting부분은 문법이 편집기에 표시되는 방법을 변경하게 한다. Element목록칸을 누르고 한개 요소를 선택한다. Family목록칸을 선택하여 그 요소의 서체형식을 변경한다. Size스핀칸을 통하여 서체크기를 선택한다. 서체는 대응하는 검사칸을 눌러서 Bold, Italic, 혹은 Underline로 바꿀수 있다. 사용되는 모든 서체는 Standard요소로부터 얻어지므로 모두에 사용되는 서체를 변경하려면 Standard요소를 변경한다. Color단추를 찰칵하여 Select Color대화칸을 펼친다. 매개 요소를 변경하였을 때 Preview행편집칸에서 변경을 볼수 있다.

Options부분에는 기정으로 검사되여있는 Wordwrap, Completion, Parentheses Matching검사칸들이 있다. 검사칸들을 선택하여 그것들을 해제할수 있다.

OK를 찰칵하여 Preferences대화칸에 대한 변경을 받아들인다. Cancel을 찰칵하여 변경을 보관하지 않고 대화칸을 닫는다.

4. Project대화칸

1) Add대화칸

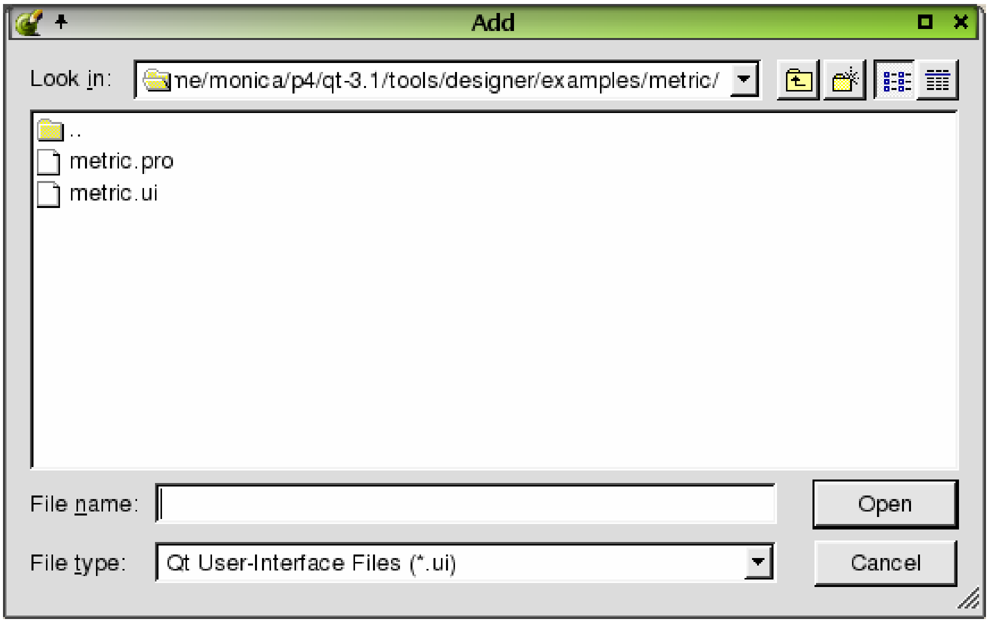


그림 1-106. Add대화칸

Project|Add File을 선택하여 Add대화칸을 펼친다. 이 대화칸을 리용하여 현재 프로젝트에 파일들을 추가한다.

Add대화칸의 기정값은 등록부와 파일형이다. 다른 등록부를 선택하려면 Look In 복합칸을 찰칵한다. 파일을 선택하면 그 이름이 File Name복합칸에 나타난다. File Type복합칸을 리용하여 다른 파일형을 선택한다. Create New Folder도구띠단추를 찰칵하여 새 등록부를 창조한다. List View도구띠단추를 찰칵하여 폴더와 파일들의 이름만 목록에 표시한다. Details도구띠단추를 찰칵하여 폴더와 파일이름, 그리고 그 크기, 형, 날자, 속성들을 표시한다. Size, Type, Date, 혹은 Attributes렬제목을 선택하여 폴더나 파일들을 정렬한다.

Open을 찰칵하여 선택한 파일을 연다. Cancel을 찰칵하여 파일을 열지 않고 대화칸을 닫는다.

2) Manage Image Collection대화칸

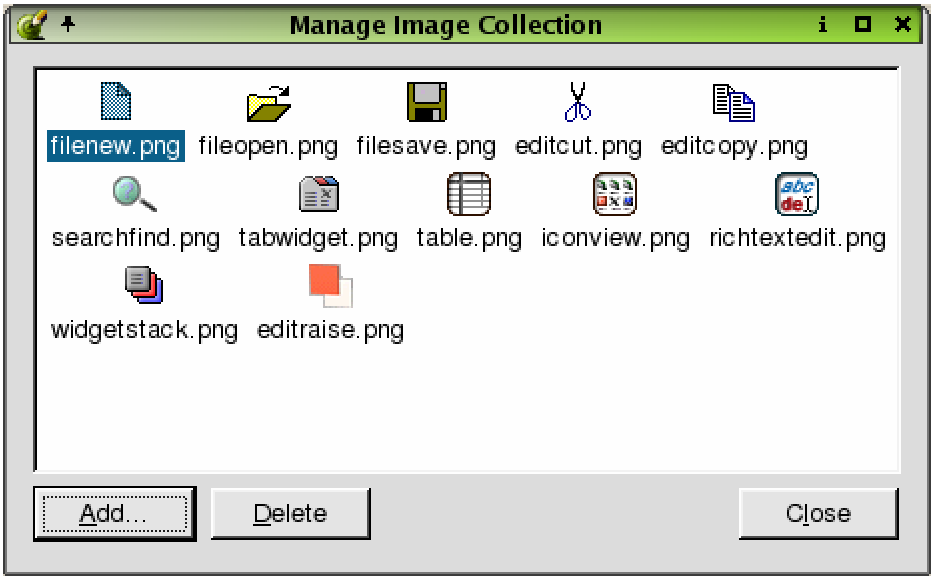


그림 1-107. Manage Image Collection대화칸

Project|Image Collection을 선택하여 Manage Image Collection대화칸을 펼친다. 이 대화칸을 리용하여 프로젝트의 화상들을 표시하고 새 화상들을 추가하거나 화상들을 삭제한다.

화상을 추가하려면 Add단추를 찰칵하여 Choose Images...대화칸을 펼친다.

Choose Images대화칸은 현재등록부와 기정파일형을 표시한다. 다른 등록부를 선택하려면 Look In복합칸을 찰칵한다. 파일을 선택하면 그 이름이 File Name복합칸에 나타난다. 다른 파일형을 선택하려면 File Type복합칸을 찰칵한다. 다른 파일들을 선택하여 대화칸의 오른쪽에 배치된 창문에서 화상들을 미리볼수 있다. Create New Folder도구띠단추를 찰칵하여 새 등록부를 창조한다. List View도구띠단추를 찰칵하여 폴더와 파일들의 이름만 목록에 표시한다. Details도구띠단추를 찰칵하여 폴더와 파일이름, 그리고 그 크기, 형, 날자, 속성들을 표시한다. Size, Type, Date, 혹은 Attributes렬제목을 선택하여 폴더나 파일들을 정렬한다. Open을 선택하여 선택된 파일을 연다. Cancel을 찰칵하여 파일을 열지 않고 대화칸을 닫는다.

그림기호보기로부터 화상을 삭제하려면 화상을 선택하고 Delete단추를 찰칵한다.

화상집합에 대한 변경이 즉시 적용된다. Close단추를 찰칵하여 대화칸을 닫는다.

3) Edit Database Connections대화칸

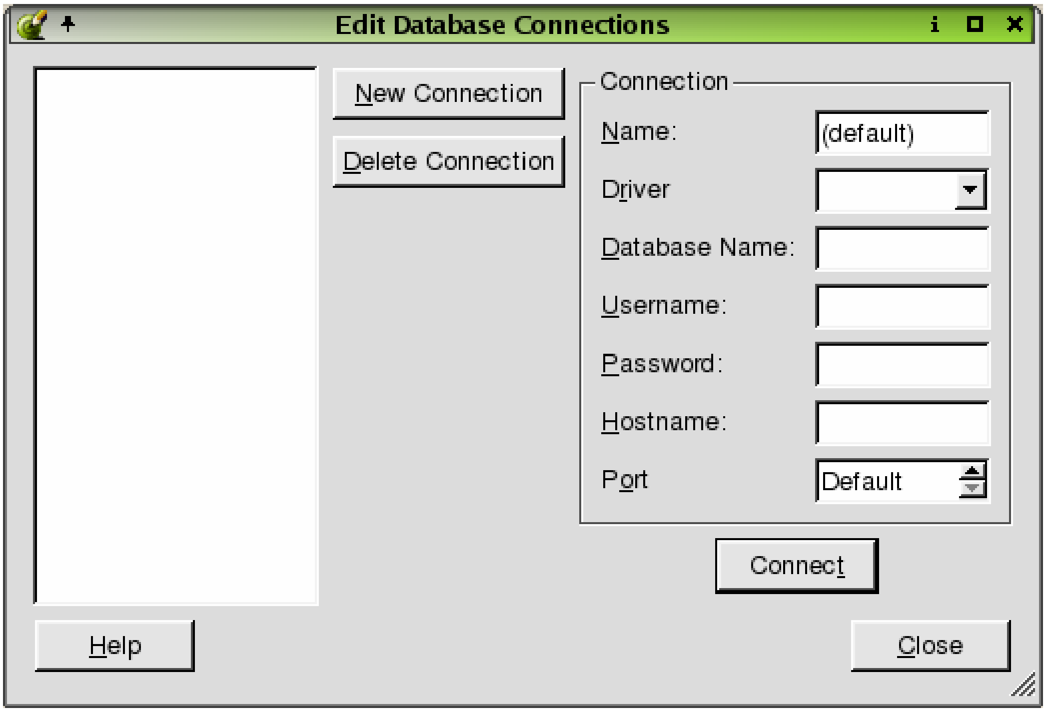


그림 1-108. Edit Database Connections대화칸

Project|Database Connections를 선택하여 Edit Database Connections대화칸을 펼친다. 이 대화칸을 리용하여 자기의 프로젝트를 자료기지에 련결하거나 현재 련결을 편집한다.

New Connection을 선택하여 새로운 자료기지련결을 창조한다. 하나의 자료기지를 사용하는 응용프로그람들에서는 기정련결이름(default)를 사용하는것이 아마 관례로 되여있다. 하나이상의 자료기지를 사용한다면 매개에 유일이름이 주어져야 한다. 구동프로그람은 Driver복합칸에서 선택되여야 한다. 자료기지이름은 Database Name복합칸에서 선택하거나 입력할수 있다. 자료기지이름, 사용자이름, 통과암호와 호스트이름은 자료기지체계관리자에 의해 제공되여야 한다. Connection정보가 끝나면 Connect를 선택한다. 련결이 이루어지면 련결이름이 대화칸의 왼쪽에 있는 목록칸에 나타난다.

련결을 삭제하려면 목록칸안의 련결을 선택하고 Delete Connection단추를 찰칵한다.

Close를 선택하여 Database Connections대화칸을 닫는다.

4) Project Settings대화칸

Project|Project Settings를 선택하여 Project Settings대화칸을 펼친다. 이 대화칸을 리용하여 프로젝트설정을 변경한다.

- Settings타브

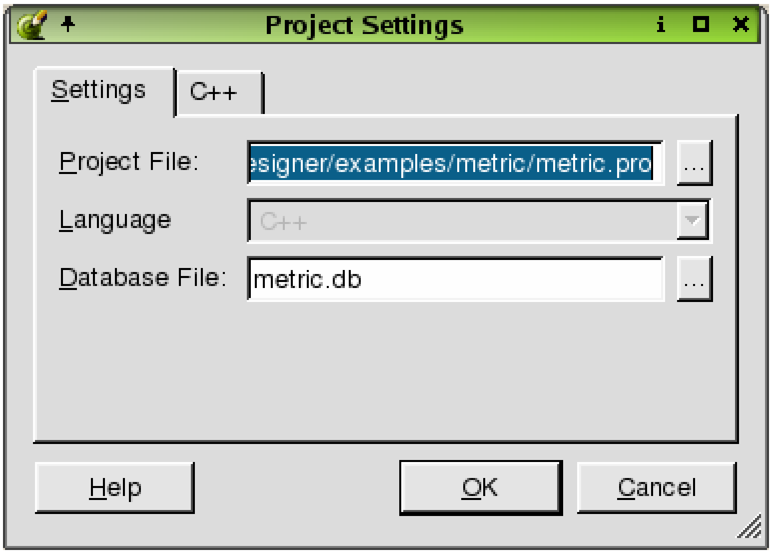


그림 1-109. Project Settings- Settings타브

Settings타브는 프로젝트에 대한 정보를 표시한다. Project File행편집칸은 기정으로 프로젝트이름이다. 이름을 변경하려면 행편집칸에 새 이름을 입력하거나 Project File열에 배치된 생략단추를 찰칵하여 Save As대화칸을 펼친다. Language복합칸은 비표시상태로 되여있다. Database File행편집칸의 이름을 변경하려면 행편집칸에 새 이름을 입력하거나 생략단추를 찰칵하여 Save As대화칸을 펼친다.

- C++타브

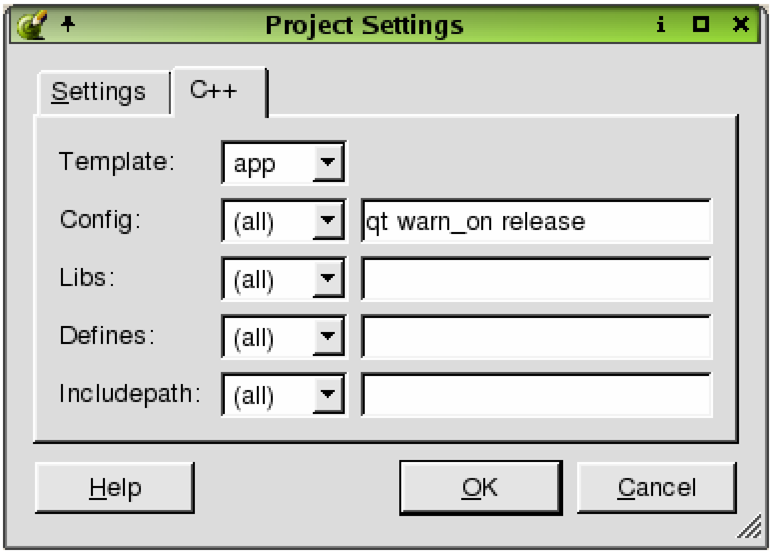


그림 1-110. Project Settings- C++타브

C++타브를 선택하여 qmake선택을 변경한다(4장 참고). Template복합칸을 리용하여 응용프로그람이나 서고를 선택하고 응용프로그람이나 서고를 건설하기 위한

makefile들을 창조한다. Config복합칸에서 모든 가동환경 혹은 특정한 가동환경용의 프로젝트환경과 콤파일러선택을 설정한다. 행편집칸에서 Config값을 입력한다.

알아두기: Unix에서는 기정으로 공유서고로 되여있다. Windows에서 공유서고를 요구한다면 기정환경지령의 끝에 dll이라고 입력하고 Template복합칸에서 lib를 선택하게 한다. Libs복합칸에서 가동환경을 선택한다. 행편집칸에 서고들을 입력한다. Defines복합칸에서 가동환경을 선택한다. Defines값들은 콤파일러의 앞처리프로그람 마크로로서 추가된다. 행편집칸에서 Defines값을 입력한다. Includepath복합칸에서 가동환경을 선택한다. Includepath는 프로젝트를 콤파일할 때 머리부파일들을 탐색하여야하는 등록부들을 지정한다. 행편집칸에서 Includepath값들을 입력한다.

OK를 찰칵하여 프로젝트설정에 대한 변경을 받아들인다. Cancel을 찰칵하여 프로젝트설정을 변경하지 않고 대화칸을 닫는다.

5. Search대화칸

1) Find Text대화칸

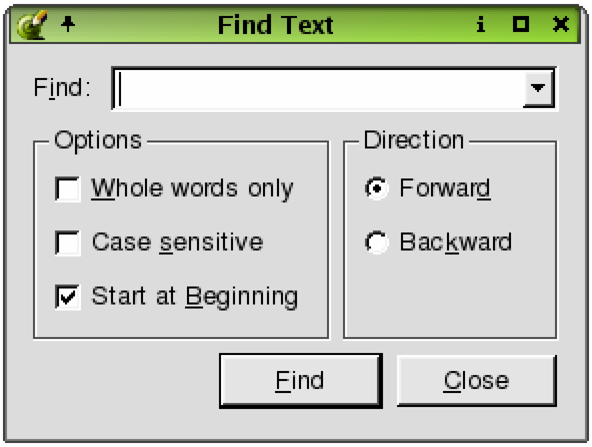


그림 1-111. Find Text

Search|Find을 선택하여(Ctrl+F을 눌러서) Find Text대화칸을 펼친다. 이 대화칸을 리용하여 프로젝트파일에서 지정된 본문을 탐색한다.

파일에서 요구되는 본문을 찾기 위하여 Find복합칸에 본문을 입력한다. Options절에서 임의의 한개 검사칸 혹은 모든 검사칸을 설정하여 탐색을 더욱 전문화할수 있다. Whole words only검사칸을 선택하여 전체 단어의 탐색을 제한한다. Case Sensitive를 선택하여 복합칸에 입력한 본문과 동등한 본문을 탐색한다. Start at Beginning을 선택하여 파일의 선두로부터 탐색을 시작한다. Direction절은 Forward라지오단추와 Backward라지오단추를 제공하여 파일에서 탐색을 진행하려는 방향을 지정한다. Find단추를 찰칵하여 탐색을 시작한다. 본문을 발견하면 그것은 파일에서 강조표시된다. Find를 계속 눌러서 련이어 놓여있는 탐색본문을 찾는다.

Close단추를 찰칵하여 대화칸을 닫는다.

2) Replace Text대화칸

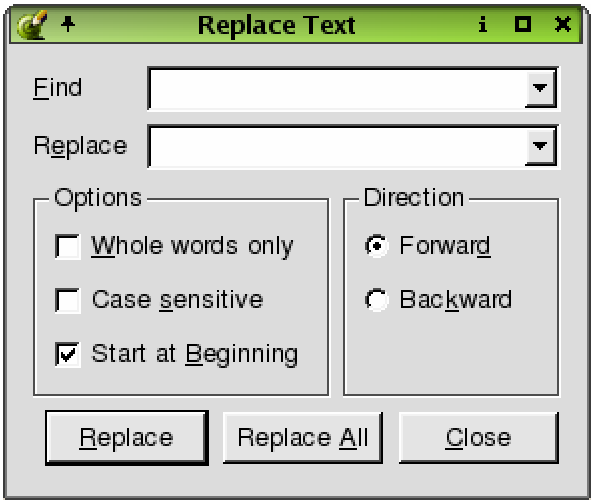


그림 1-112. Replace Text

Search|Replace을 선택하여(또는 Ctrl+R를 눌러서) Replace Text대화칸을 펼친다. 이 대화칸을 리용하여 프로젝트파일안의 본문을 교체한다.

본문을 교체하려면 Find복합칸에 교체하려는 본문을 입력한다. Replace복합칸에 새 본문을 입력한다. Options절의 검사칸들을 일부 혹은 전부 선택하여 탐색방법을 선택할수도 있다. Whole words only검사칸을 선택하여 전체 단어에 대한 탐색을 제한할수 있다. Case Sensitive를 선택하여 복합칸에 입력한 본문과 등가한 본문을 탐색한다. Start at Beginning을 선택하여 파일의 선두로부터 탐색을 시작한다. Direction절은 Forward라지오단추와 Backward라지오단추를 제공하여 파일에서 탐색을 수행하는 방향을 지정한다.

Replace단추를 찰칵하여 본문을 탐색 및 교체한다. 본문이 발견될 때 파일안에서 선택표시된다. Replace단추를 찰칵하여 파일안에서 그 본문이 발생할 때마다 탐색과 교체를 계속한다. Replace All단추를 찰칵하여 파일에서 모든 탐색본문을 한번에 교체한다.

Close단추를 찰칵하여 대화칸을 닫는다.

3) Goto Line대화칸



그림 1-113. Goto Line

Search|Goto line을 선택하여(혹은 Alt+G를 눌러서) Goto Line대화칸을 펼친다.

이 대화칸을 리용하여 파일안의 지정행으로 이행한다.

행번호를 선택하려면 Line스핀칸에 번호를 입력하거나 스핀칸에서 올리와 내리화살표를 찰칵한다. Goto단추를 찰칵한다. 유표는 파일의 행선두에 놓인다.

Close단추를 찰칵하여 대화칸을 닫는다.

6. Tools대화칸

1) Edit Custom Widgets대화칸

Edit Custom Widgets대화칸은 Tools|Custom|Edit Custom Widgets를 선택하여 호출된다. 이 대화칸을 리용하여 사용자정의창문부품을 창조한다.

사용자정의창문부품은 코드로 창조된다. 사용자정의창문부품들은 현존창문부품들의 결합을 포함하지만 기능과 처리부, 신호들이 추가되여있다. 혹은 무로부터 사용자정의창문부품을 쓸수도 있으며 두개를 결합하여 만들수도 있다. 사용자정의창문부품은 흔히 다른 창문부품의 특수화(파생클라스) 혹은 함께 작업하는 창문부품들의 결합 혹은 이 두가지 수법의 결합이다. 특별한 환경에서 단지 창문부품들의 집합을 바란다면 그것들을 창조하는것은 가장 간단하며 그것들을 그룹으로 선택하고 Qt Designer에서 요구될 때 그것들을 복사하여 붙이기한다. 사용자정의창문부품들은 일반적으로 현존창문부품 혹은 창문부품그룹에 새 기능을 추가하여야 할 때 창조된다. 새 창문부품을 추가하려면 New Widget단추를 찰칵한다. Definitions Section에서 새 창문부품추가에 대한 더 자세한 정보를 볼수 있다. 사용자정의창문부품서술을 포함하는 파일을 적재하려면 Load Descriptions단추를 찰칵하여 Open대화칸을 연다. 렬거된 사용자정의창문부품들의 서술을 보관하려면 Save Descriptions단추를 찰칵하여 Save As대화칸을 펼친다. 창문부품을 삭제하려면 목록칸에서 그 창문부품을 선택하고 Delete Widget단추를 찰칵한다.

Close를 선택하여 Edit Custom Widgets대화칸을 닫는다.

- Definition타브

사용자정의창문부품을 창조하려면 New Widget를 선택한다.(그림 1-114) 이미 그것이 열려 있으면 Definition타브를 선택한다. 행편집칸에 입력하여 Class이름을 MyCustomWidget로부터 유일이름으로 변경해야 한다. Headerfile행편집칸에 입력하여 이름을 변경하거나 리용하려는 머리부파일의 이름을 입력한다. 등록부에서 보관된 머리부파일을 탐색하려면 Headerfile행편집칸의 오른쪽에 있는 생략단추를 찰칵하여 Open대화칸을 펼친다. Select Access복합칸에서 파일을 포함하는 방법을 선택한다. 대역머리부파일은 각괄호(<>)를 리용하여 포함한다. 국부파일은 2중인용표를 리용하여 포함한다. 도구띠에서 자기 창문부품을 식별하는데 리용하려는 픽스매프가 있으면 Pixmap표식자의 오른쪽에 있는 생략단추를 찰칵한다. 이것은 Choose a Pixmap대화칸을 펼친다. Size Hint스핀칸들에서 요구되는 창문부품크기를 선택한다. 요구되는 크기를 사용하지 않으려고 한다면 스핀칸들에 -1/-1을 입력한다. Size Policy복합칸들에서 창문부품의 수직크기속성들을 선택한다. 자기가 창조한 사용자정의창문부품이 다른 창문부품(자식)들을 포함할수 있어야 한다면 Container Widget검사칸을 선택한다.

- Signals타브

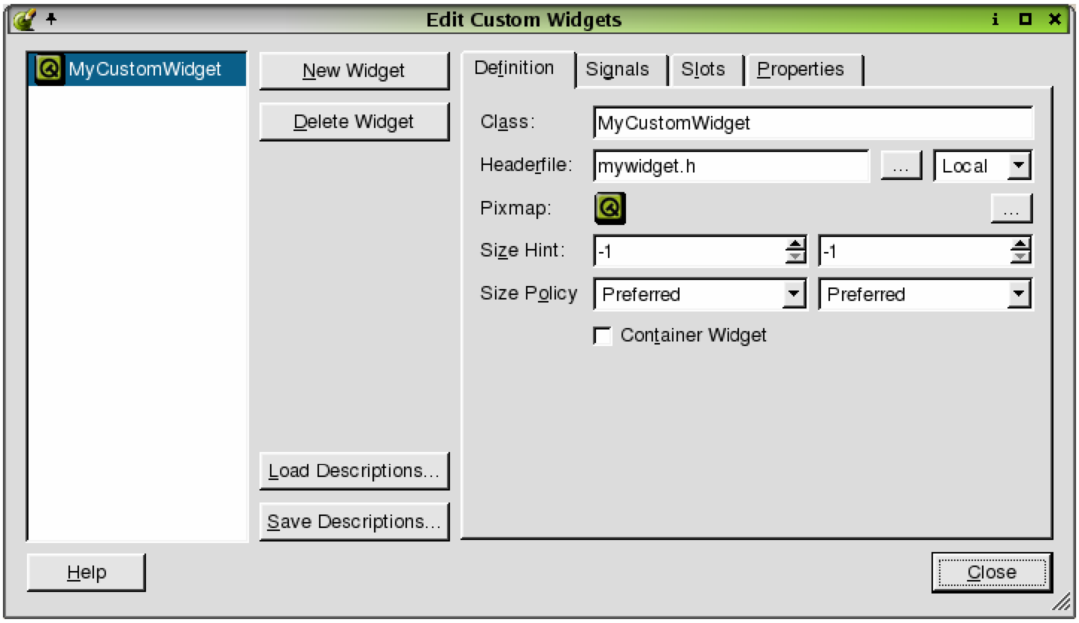


그림 1-114. Edit Custom Widgets- Definition타브

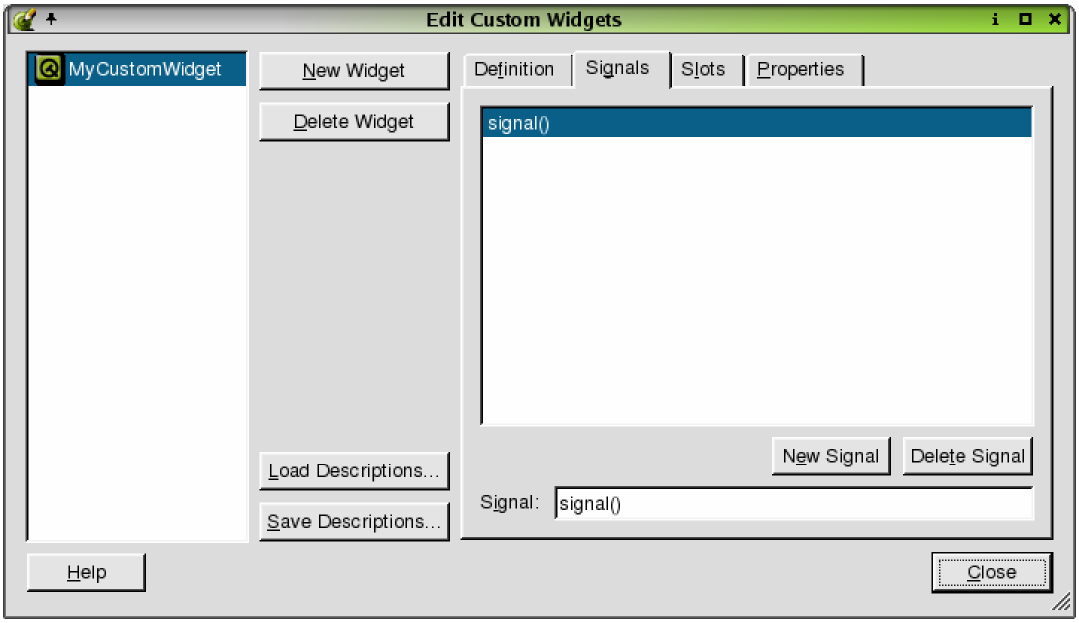


그림 1-115. Edit Custom Widgets- Signals타브

Signals타브를 선택하여 선택된 사용자정의창문부품이 발생할수 있는 모든 신호목록을 표시할수 있다. 새 신호를 추가하려면 New Signal단추를 찰칵한다. Signal행편집칸을 선택하고 신호의 인수를 제공하고 유일이름을 준다. 목록칸에서 신호를 삭제하려면 그 신호를 선택하고 Delete Signal단추를 찰칵한다.

- Slots타브

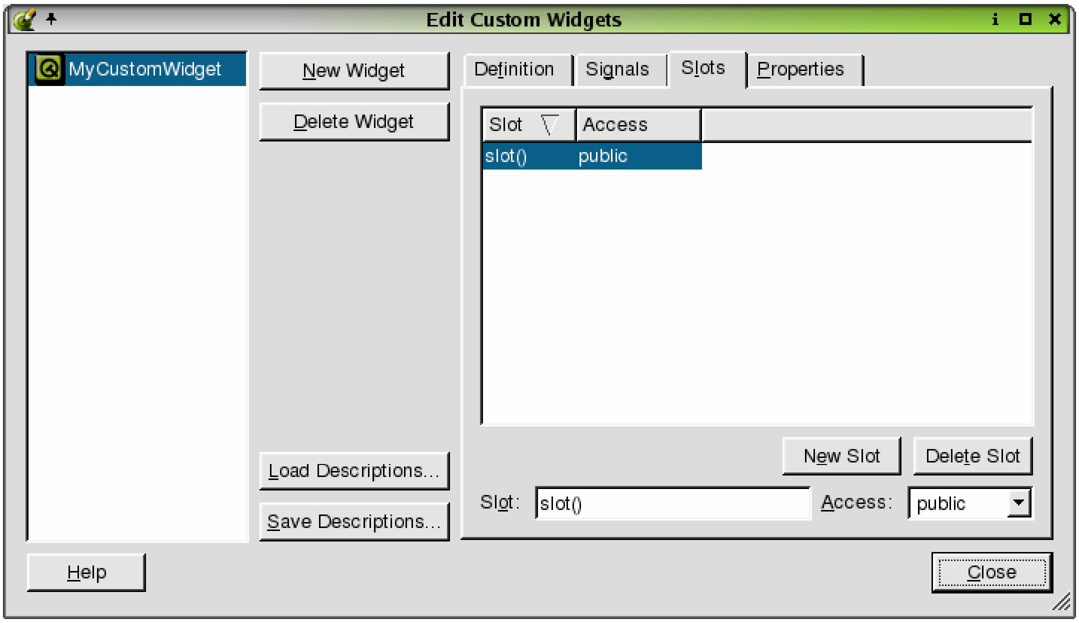


그림 1-116. Edit Custom Widgets- Slots타브

Slots타브를 선택하여 선택된 사용자정의창문부품용의 모든 처리부들의 목록을 표시한다. Slot나 Access렬제목을 선택하여 목록칸안의 처리부들을 정렬한다. 처리부를 추가하려면 New Slot단추를 찰칵한다. Slot행편집칸을 선택하고 처리부용인수를 제공하고 처리부에 유일이름을 준다. Access복합칸에서 창문부품에 대하여 공개 혹은 보호호출을 선택한다. 목록칸에서 처리부를 삭제하려면 그 처리부를 선택하고 Delete Slot을 찰칵한다.

- Properties타브

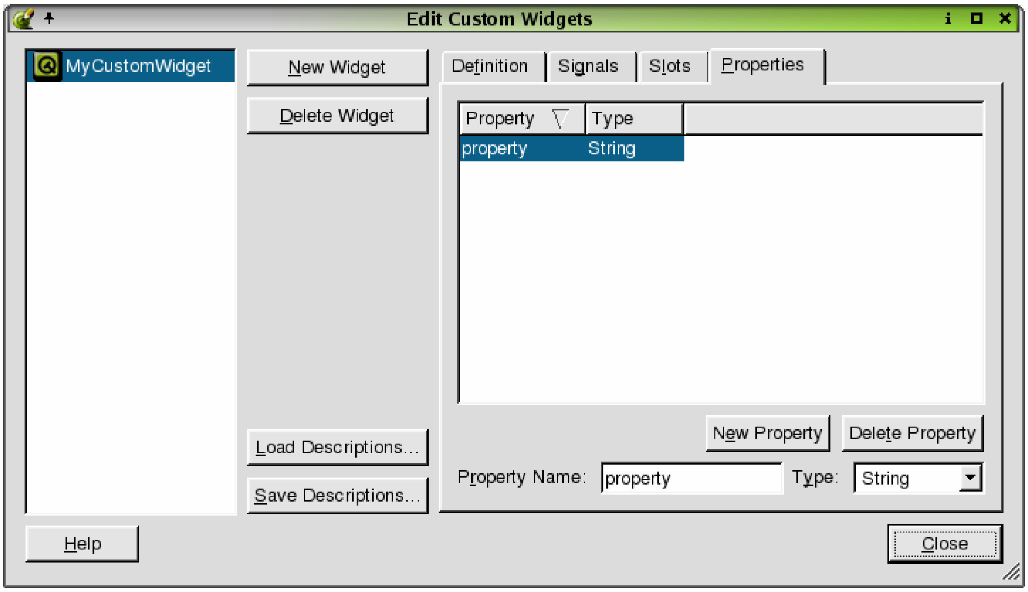


그림 1-117. Edit Custom Widgets- Properties타브

Properties타브를 선택하여 선택된 창문부품의 속성목록을 표시한다. Property나 Type렬제목을 선택하여 목록칸안의 속성들을 정렬한다. 속성을 추가하려면 New Property단추를 찰칵한다. 속성의 기정이름을 변경하려면 Property Name행편집칸을 선택한다. Qt의 속성체계를 리용하여 클라스에 속성들을 실현하여야 한다. 속성형을 선택하려면 Type복합칸을 찰칵한다. 목록칸에서 속성을 삭제하려면 그 속성을 선택하고 Delete Property단추를 찰칵한다.

Close를 선택하여 Edit Custom Widgets 대화칸을 닫는다.

2) Configure Toolbox대화칸

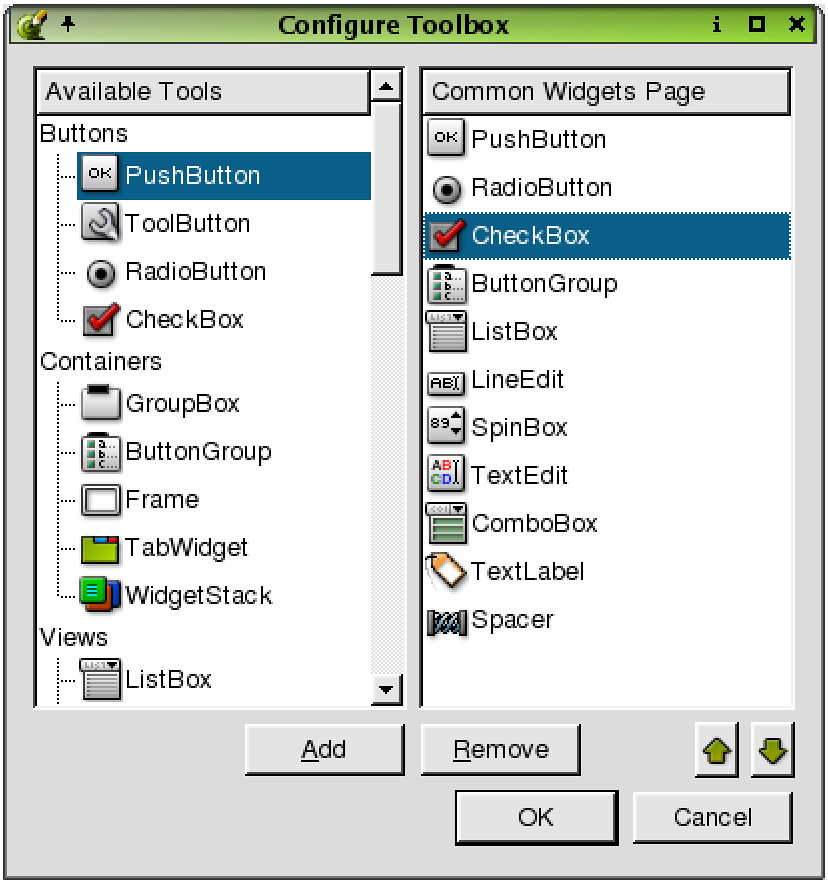


그림 1-118. Configure Toolbox

Tools|Configure Toolbox를 선택하여 Configure Toolbox대화칸을 펼친다. 이 대화칸은 유효도구목록과 도구칸에서 Common Widgets범주의 도구목록을 표시하는데 리용된다. Common Widgets범주에 창문부품들을 추가하려면 Available Widgets목록에서 창문부품을 선택하고 Add를 찰칵한다. 선택된 창문부품이 현재 Common Widgets범주에 나타난다. 창문부품을 Common Widgets목록에서 삭제하려면 창문부품을 선택하고 Remove를 찰칵한다. Common Widgets목록에서 창문부품을 우, 아래로

이동하려면 올리 및 내리화살표들을 찰칵한다.

OK를 찰칵하여 도구칸에 환경구성을 받아들인다. Cancel을 찰칵하여 대화칸을 닫는다.

7. Help대화칸

1) About Qt Designer대화칸

Help|About를 선택하여 About Qt Designer 대화칸을 펼친다. 이 대화칸은 판, 허가항목, 조건, 거부자와 같은 Qt Designer에 대한 정보를 제공한다.

대화칸의 오른쪽웃구석에 배치되여있는 X를 선택하여 대화칸을 닫는다.

2) About Qt대화칸

Help|About Qt를 선택하여 About Qt대화칸을 펼친다. 이 대화칸은 Qt에 대한 정보를 제공한다.

대화칸의 제목에 배치되여있는 X를 선택하여 대화칸을 닫는다.

8. Widget대화칸

1) Edit Text대화칸

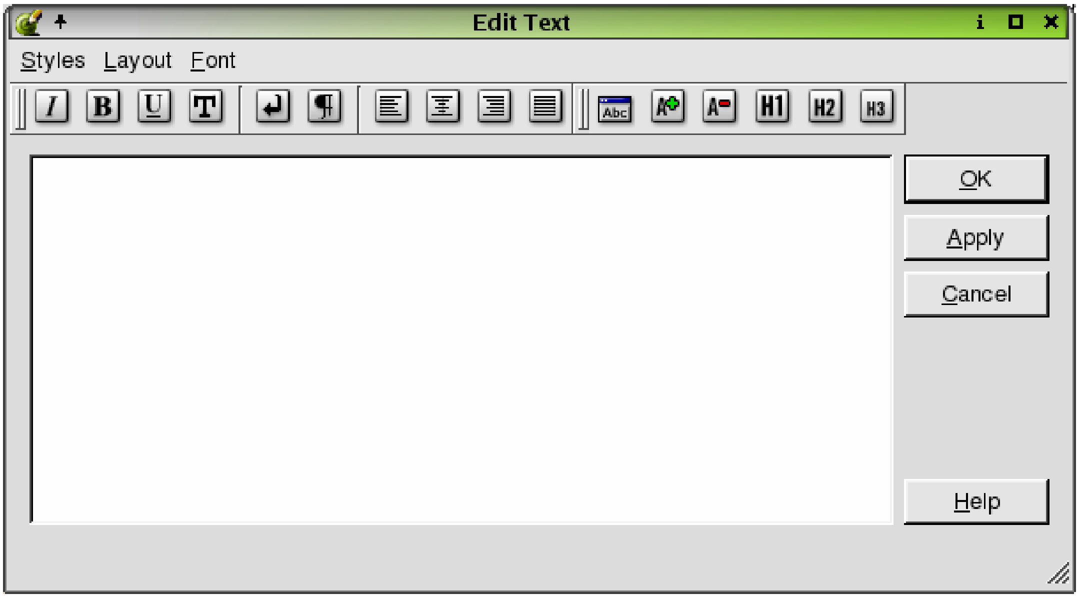


그림 1-119. Edit Text

Edit Text대화칸은 다음의 창문부품들 즉 TextEdit, TextLabel, PixmapLabel를 오른쪽 단추로 선택하고 Edit Text를 눌러서 펼친다.

이 대화칸을 리용하여 형식도구띠와 차림표를 리용하는 자기의 본문에 HTML형식화를 적용한다.

OK를 찰칵하여 본문과 형식화를 받아들인다. Cancel을 찰칵하여 본문이나 형식화를 보관하지 않고 대화칸을 닫는다.

2) Edit Text대화칸

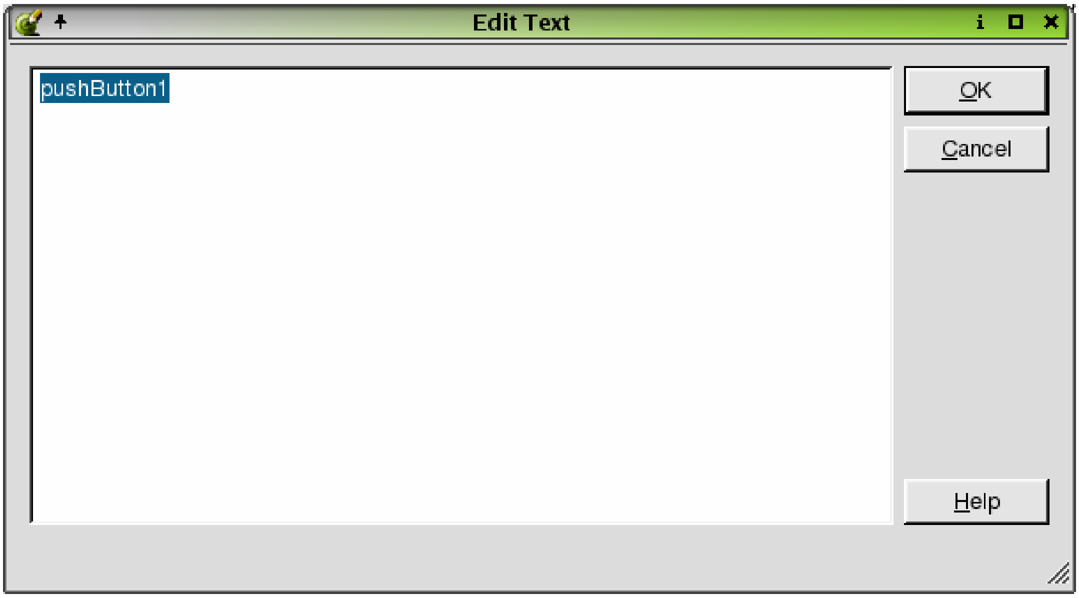


그림 1-120. Edit Text

Edit Text대화칸은 다음의 창문부품들 즉 PushButton, RadioButton, CheckBox, ToolButton을 오른쪽 단추로 선택하고 Edit Text를 눌러서 펼친다.

이 대화칸을 리용하여 창문부품의 본문을 변경한다.

OK를 찰칵하여 본문에 대한 변경을 받아들인다. Cancel을 찰칵하여 본문에 대한 변경을 보관하지 않고 대화칸을 닫는다.

3) Text대화칸

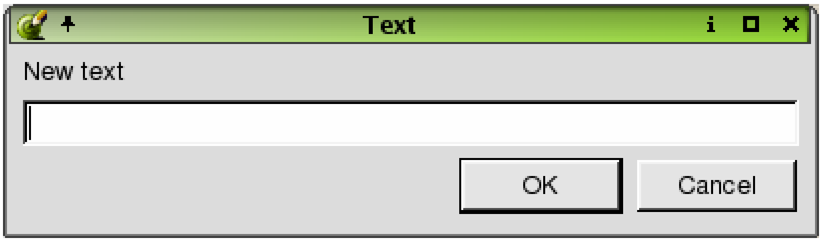


그림 1-121. Text대화칸

Text대화칸은 LineEdit창문부품을 오른쪽 단추로 눌러서 펼친다.

이 대화칸은 행편집칸의 본문을 변경하는데 리용된다.

OK를 찰칵하여 본문에 대한 변경을 받아들인다. Cancel을 찰칵하여 본문에 대한 변경을 보관하지 않고 대화칸을 닫는다.

4) Title대화칸

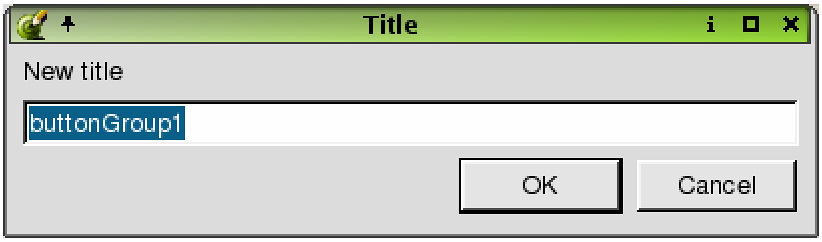


그림 1-122. Title대화칸

Title대화칸은 다음의 창문부품들 즉 ButtonGroup과 GroupBox를 오른쪽 단추로 선택하고 Edit Title를 눌러서 펼친다.

이 대화칸을 리용하여 행편집칸에 새로운 제목을 입력하여 선택된 창문부품의 제목을 변경한다.

OK를 찰칵하여 제목에 대한 변경을 받아들인다. Cancel을 찰칵하여 제목을 변경하지 않고 대화칸을 닫는다.

5) Page Title대화칸



그림 1-123. Page Title대화칸

Page Title대화칸 TabWidget를 오른쪽 단추로 선택하고 Edit Page Title를 눌러서 펼친다.

이 대화칸을 리용하여 Tab창문부품안의 매개 타브의 제목을 변경한다.

OK를 찰칵하여 새 페지제목을 받아들인다. Cancel을 찰칵하여 변경하지 않고 대화칸을 닫는다.

6) Choose an Image대화칸

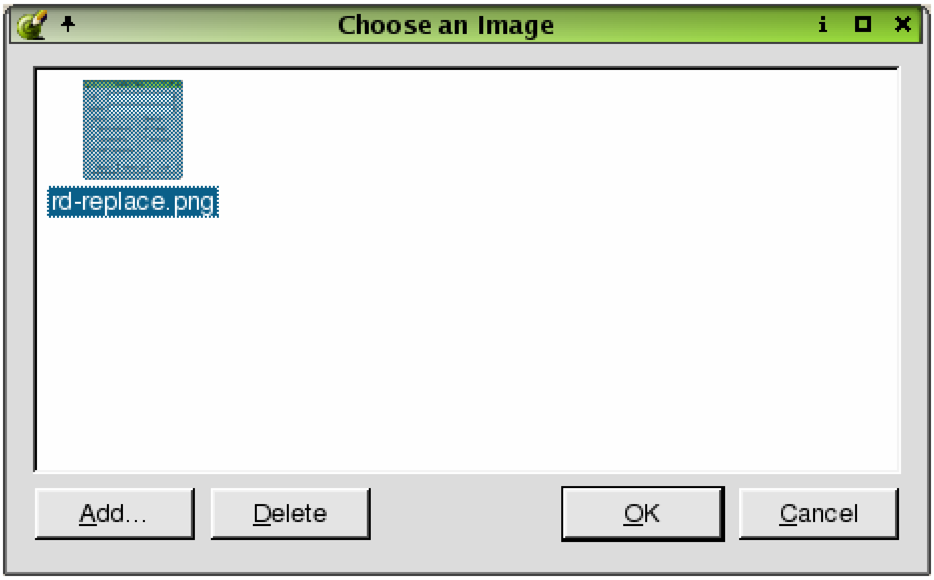


그림 1-124. Choose an Image

Choose an Image대화칸에서 창문부품에서 리용하려는 화상을 선택한다.

목록칸에서 화상을 선택하려면 화상을 선택하고 OK를 찰칵한다. 화상을 추가하려면 Add단추를 찰칵하여 Choose Images...대화칸을 펼친다. 화상을 삭제하려면 목록칸에서 화상을 선택하고 Delete단추를 찰칵한다.

Cancel을 찰칵하여 화상을 변경하지 않고 대화칸을 닫는다.

7) Edit Listbox대화칸

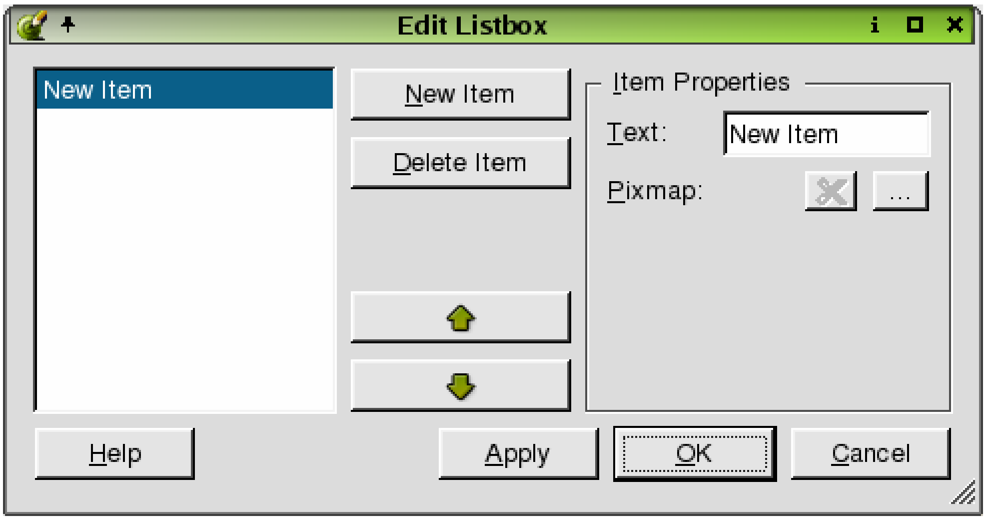


그림 1-125. Edit Listbox

Edit Listbox대화칸은 Listbox창문부품을 오른쪽 단추로 선택하고 Edit단추를 찰칵하여 기동한다.

이 대화칸을 리용하여 목록칸에 항목들을 추가하고 항목의 속성들을 변경한다.

목록칸에 항목을 추가하려면 New Item을 선택한다. 항목의 기정이름을 변경하려면 Item Properties절안의 Text행편집칸을 선택하고 항목의 새 이름을 입력한다. Select a Pixmap를 선택하여 Choose an Image대화칸을 펼친다. 픽스매프를 선택하고 Delete Pixmap단추를 찰칵하여 선택된 픽스매프를 삭제한다. 목록칸에서 항목을 삭제하려면 그 항목을 선택하고 Delete단추를 찰칵한다. 목록칸에서 항목을 우, 아래로 이동하려면 Move Up 혹은 Move Down단추를 찰칵한다. Apply를 선택하여 변경을 받아들인다.

Apply를 선택하여 목록칸 창문부품에 대한 변경을 받아들인다. 일단 변경을 받아들였으면 OK를 찰칵하여 대화칸을 닫는다. Cancel을 찰칵하여 변경을 보관하지 않고 대화칸을 닫는다.

8) Edit Listview

Edit Listview대화칸은 목록보기창문부품을 오른쪽 단추로 선택하고 Edit를 눌러서 기동한다.

이 대화칸을 리용하여 목록보기에 항목을 추가한다. Edit Listview대화칸에는 두개의 타브가 있는데 하나는 항목용이고 다른 하나는 렬용이다.

- Items타브

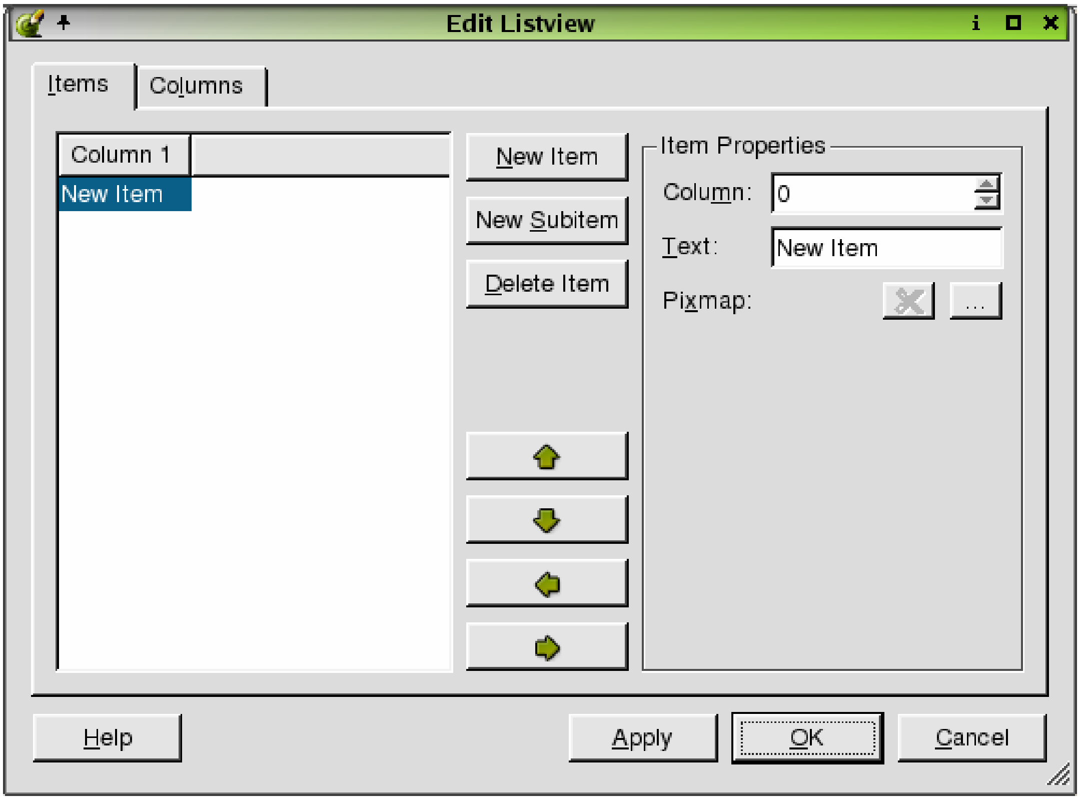


그림 1-126. Edit Listview- Items타브

대화칸은 기정으로 Items타브로 되여있다. 이 타브를 리용하여 목록보기에서 항목들을 추가, 변경 혹은 삭제한다. 새 항목을 추가하려면 New Item단추를 찰칵한다. 새 항목은 목록칸의 꼭대기에 표시된다. 현존항목에 보조항목들을 추가하려면 그 항목을 선택하고 New Subitem단추를 찰칵한다. Column스핀칸에서 항목본문 혹은 픽스매프를 배치할 렬을 선택한다. Text행편집칸을 선택하여 렬의 본문을 입력하거나 항목이나 보조항목의 이름을 변경한다. 픽스매프를 선택하고 Delete Pixmap단추를 찰칵하여 선택된 픽스매프를 삭제한다. 목록칸에서 항목을 삭제하려면 그 항목을 선택하고 Delete단추를 찰칵한다. 계층준위에서 항목을 우아래로 이동하려면 Move Up 혹은 Move Down단추를 찰칵한다. 항목을 한준위 우아래로 이동하려면 Move Left 혹은 Move Right단추를 찰칵한다.

- Columns타브

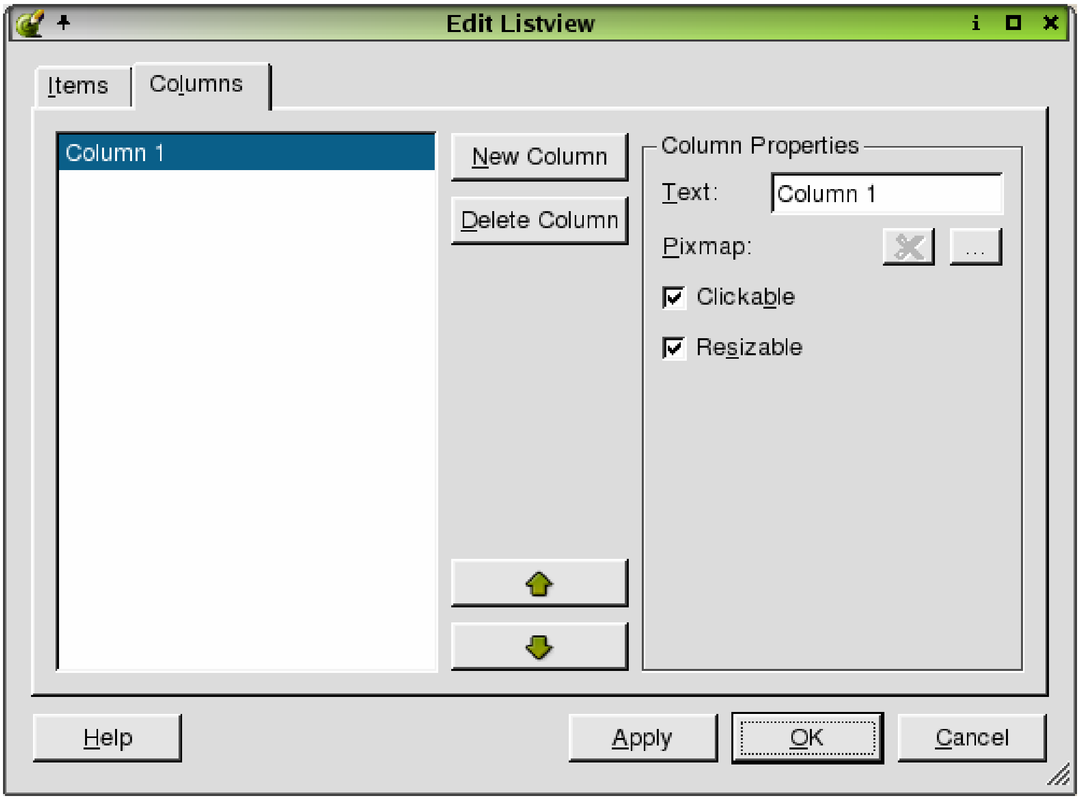


그림 1-127. Edit Listview- Columns타브

이 타브를 선택하여 목록보기의 렬구성을 변경한다. 렬을 추가하려면 New Column단추를 찰칵한다. 새 렬이 목록칸의 꼭대기에 표시된다. 렬이름을 변경하려면 목록칸안의 렬을 선택하고 Text행편집칸에서 새 이름을 입력한다. 픽스매프를 추가하려면 생략단추를 찰칵하여 Choose an Image대화칸을 펼친다. 픽스매프를 삭제하려면 Delete Pixmap단추를 찰칵한다. 렬이 마우스누르기에 응답하게 하려면 Clickable검사

칸을 선택한다. 렬의 폭을 변경할수 있게 하려면 Resizeable검사칸을 선택한다. 렬을 삭제하려면 목록칸안의 렬을 선택하고 Delete Column단추를 찰칵한다. 목록칸에서 우아래로 렬을 이동하려면 Move Up 혹은 Move Down단추를 찰칵한다.

Apply를 선택하여 목록보기 창문부품에 대한 변경을 받아들인다. 일단 변경을 받아들였으면 OK를 찰칵하여 대화칸을 닫는다. Cancel을 찰칵하여 변경을 보관하지 않고 대화칸을 닫는다.

9) Edit Iconview대화칸

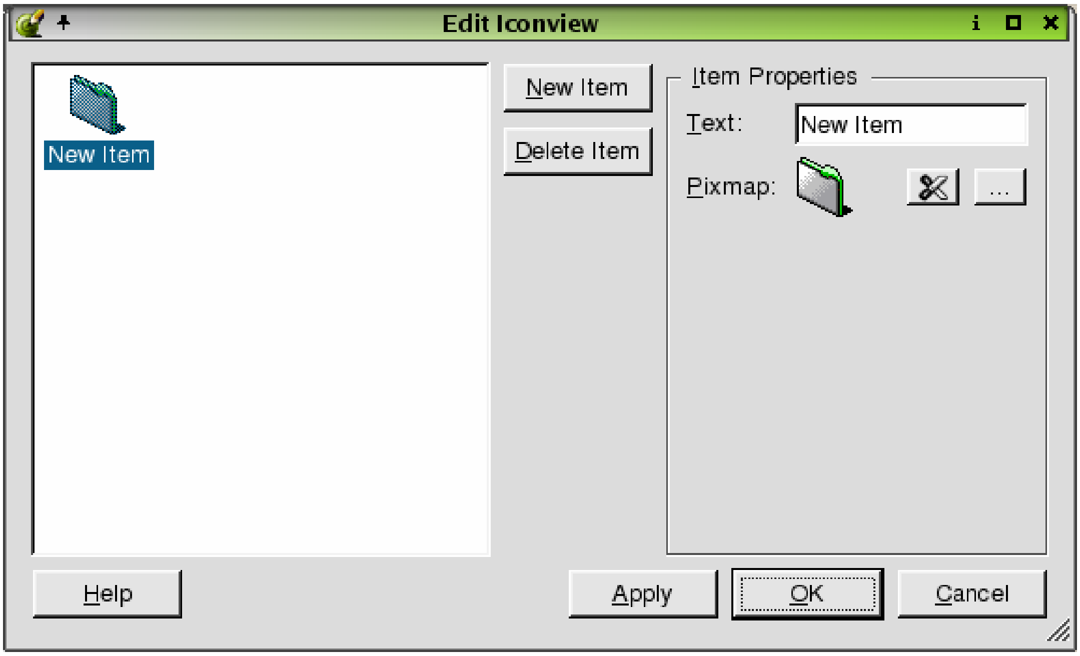


그림 1-128. Edit Iconview

Edit Iconview대화칸은 그림기호보기 창문부품을 오른쪽 단추로 선택하고 Edit.를 눌러서 펼친다.

대화칸을 리용하여 그림기호보기로부터 항목들을 추가, 변경, 혹은 삭제할수 있다. 그림기호보기에 항목을 추가하려면 New Item단추를 찰칵한다. 항목의 이름을 변경하려면 Text행편집칸을 선택하고 새 이름을 입력한다. 픽스매프를 추가하려면 생략단추를 찰칵하여 Choose an Image대화칸을 펼친다. 픽스매프를 삭제하려면 Delete Pixmap단추를 찰칵한다. 그림기호보기로부터 항목을 삭제하려면 그 항목을 선택하고 Delete Item단추를 찰칵한다.

Apply를 선택하여 그림기호보기 창문부품에 대한 변경을 받아들인다. 일단 변경을 받아들였으면 OK를 찰칵하여 대화칸을 닫는다. Cancel을 찰칵하여 변경을 보관하지 않고 대화칸을 닫는다.

10) Edit Table대화칸

Edit Table대화칸은 다음의 창문부품들 즉 Table과 DataTable을 오른쪽 단추로 선택하고 Edit를 눌러서 펼친다.

이 대화칸을 리용하여 표에서 렬이나 행들을 추가, 변경, 혹은 삭제한다.

- Column타브

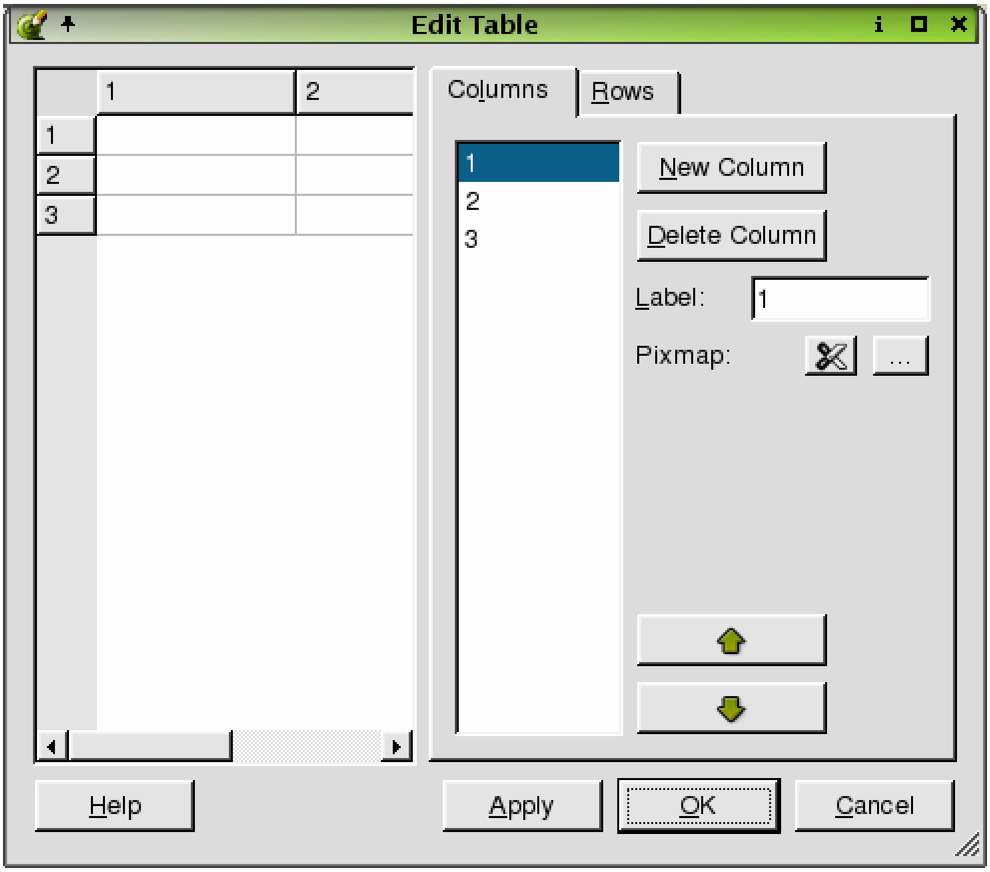


그림 1-129. Edit Table- Columns타브

표에 렬을 추가하려면 New Column단추를 찰칵한다. 표에서 렬을 삭제하려면 삭제하려는 렬을 선택하거나 Columns목록칸에서 렬번호를 선택하고 Delete Column단추를 찰칵한다. 렬이름을 변경하려면 Label행편집칸을 선택하고 다음 본문을 입력한다. 픽스매프를 추가하려면 생략단추를 찰칵하여 Choose an Image대화칸을 펼친다. 선택된 항목의 현재 렬에서 픽스매프를 삭제하려면 Delete Pixmap단추를 찰칵한다. 목록칸에서 렬을 이동하려면 Move Up 혹은 Move Down단추를 찰칵한다.

- Rows타브

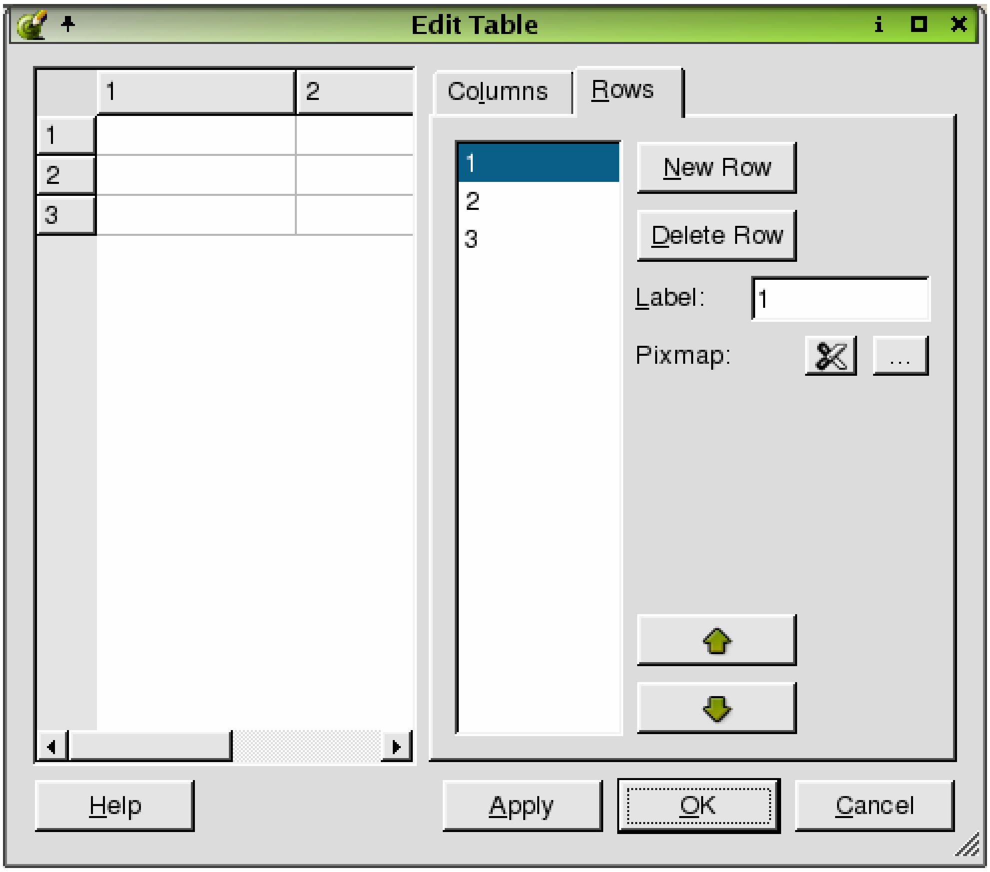


그림 1-130. Edit Table- Rows타브

표에 렬을 추가하려면 New Row단추를 찰칵한다. 표에서 행을 삭제하려면 표에서 삭제하려는 행을 선택하거나 Rows목록칸에서 행번호를 선택하고 Delete Column단추를 찰칵한다. 행이름을 변경하려면 행이나 행번호를 선택하고 Label행편집칸에서 다음 본문을 입력한다. 픽스매프를 추가하려면 생략단추를 찰칵하여 Choose an Image대화칸을 펼친다. 선택된 항목의 현재행으로부터 픽스매프를 삭제하려면 Delete Pixmap단추를 찰칵한다. 목록칸의 한 행을 이동하려면 Move Up 혹은 Move Down단추들을 찰칵한다.

Apply를 선택하여 표창문부품에 대한 변경을 받아들인다. 일단 변경을 받아들였으면 OK를 찰칵하여 대화칸을 닫는다. Cancel을 찰칵하여 변경을 보관하지 않고 대화칸을 닫는다.

9. Property Editor대화칸

1) Choose a Pixmap대화칸

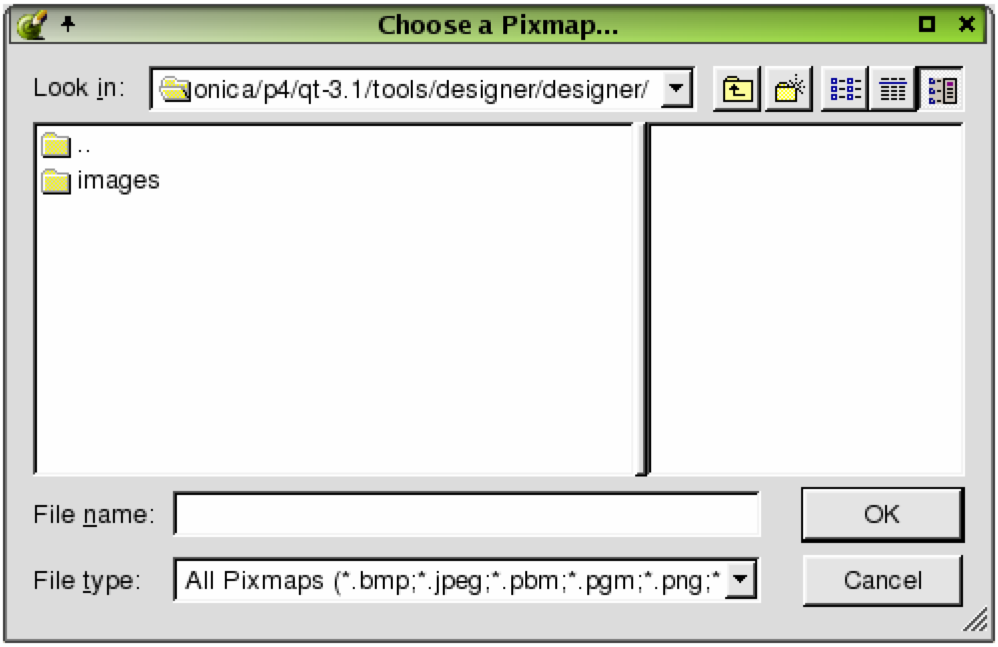


그림 1-131. Choose a Pixmap

Choose a Pixmap대화칸은 paletteBackgroundPixmap 혹은 속성편집기의 픽스매프속성열의 단추를 찰칵하여 펼친다. 또한 다음과 같은 창문부품들 즉 PushButton, RadioButton, CheckBox, ToolButton을 오른쪽 단추로 선택하여 펼칠수도 있다. 또한 이 대화칸은 다음의 창문부품들 즉 Table, DataTable, ListBox, ComboBox, ListView, IconView를 오른쪽 단추로 선택하고 Edit를 누른 다음 Select a Pixmap단추를 찰칵하여 호출할수도 있다.

이 대화칸을 리용하여 현재 프로젝트에서 사용하려는 픽스매프를 선택한다.

Choose a Pixmap대화칸은 현재등록부와 기정파일형을 표시한다. 다른 등록부를 선택하려면 Look In복합칸을 찰칵한다. 파일을 선택하면 그 이름이 File Name복합칸에 나타난다. 다른 파일형을 선택하려면 File Type복합칸을 찰칵한다. Create New Folder도구띠단추를 찰칵하여 새 등록부를 창조한다. List View도구띠단추를 찰칵하여 폴더와 파일들의 이름만 목록에 표시한다. Details도구띠단추를 찰칵하여 폴더와 파일이름, 그리고 그 크기, 형, 날자, 속성들을 표시한다. Size, Type, Date 혹은 Attributes렬제목을 선택하여 폴더나 파일들을 정렬한다. 대화칸의 오른쪽에 배치된 미리보기칸에서 선택하는 픽스매프의 견본을 표시한다.

OK를 찰칵하여 픽스매프파일을 받아들인다. Cancel을 찰칵하여 픽스매프파일을 선택하지 않고 대화칸을 닫는다.

2) Select Color대화칸

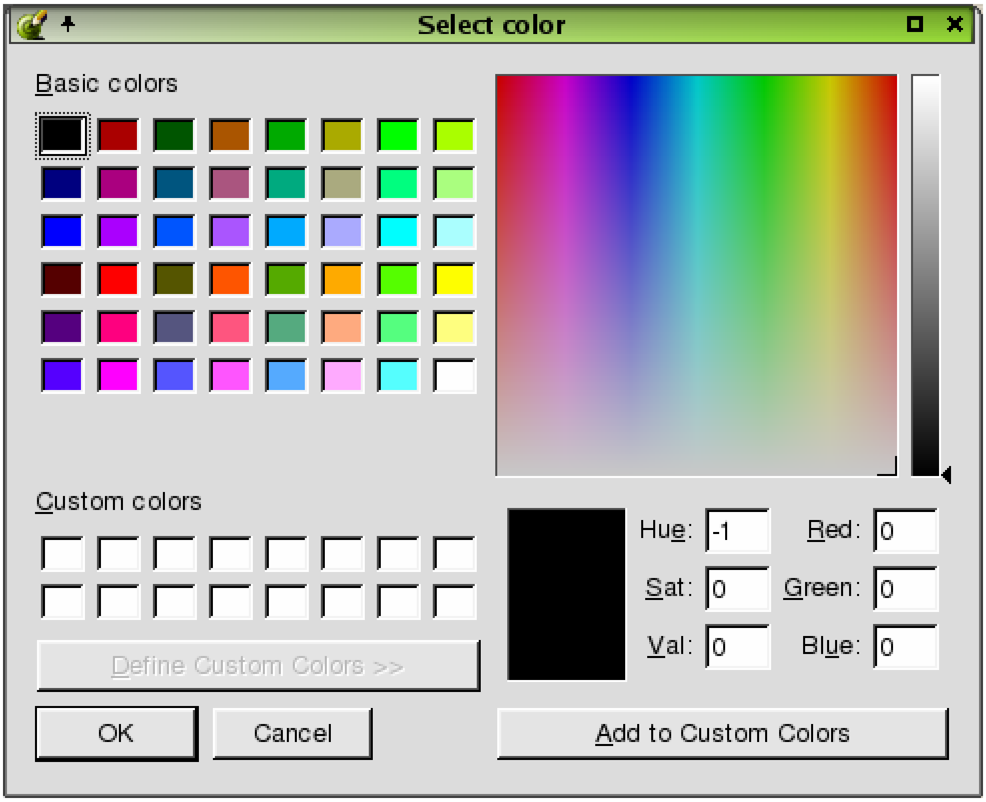


그림 1-132. Select Color

Select Color대화칸은 paletteForegroundColor열의 단추나 속성편집기의 paletteBackgroundColor속성열의 단추들을 눌러서 펼친다.

이 대화칸을 리용하여 색을 선택하거나 조색판을 창조한다.

Basic Colors절로부터 색을 선택하면 색견본이 대화칸의 바닥에 있는 작은 미리보기칸에 나타난다. 색견본의 오른쪽에 색스펙트르안에 색의 위치정보가 있는 행편집칸들을 보게 된다. 또한 더 큰 색스펙트르창문안의 +기호는 색의 위치를 보여준다. 또한 전용색들의 조색판을 창조할수도 있다. 전용색을 추가하는 두가지 방법이 있다. 색스펙트르창문의 색을 선택한다. 창문아래의 작은 칸에 색이 나타날 때 색을 선택하고 그것을 대화칸의 Custom Color절안에서 빈칸들중 하나로 끌고간다. 또한 Basic Colors절로부터 색들을 선택하여 끌고다닐수 있다. 색을 추가하는 다른 방법은 색이 선택되였을 때 Add to Custom Colors를 누르는것이다.

OK를 찰칵하여 Select Color 대화칸에 대한 변경을 받아들인다. Cancel을 찰칵하여 색을 선택하지 않고 혹은 전용색들을 추가하지 않고 대화칸을 완료한다.

3) Edit Palette대화칸

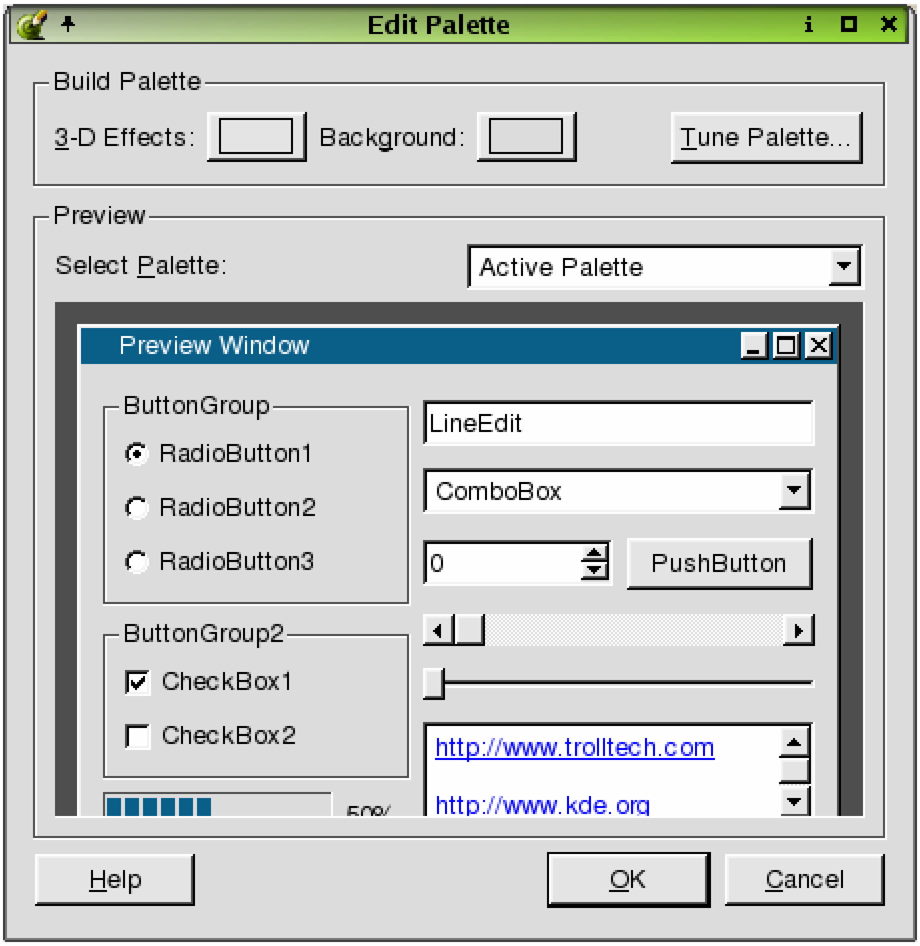


그림 1-133. Edit Palette

Edit Palette대화칸은 속성편집기의 조색판속성열의 단추를 찰칵하여 펼친다.

이 대화칸을 리용하여 현재 창문부품이나 폼의 조색판을 변경한다. 생성한 조색판을 리용하거나 각 색그룹과 각 색의 역할을 선택할수 있다. 조색판은 미리보기절에서 각이한 창문부품배치에 대하여 시험할수 있다.

Build Palette절은 조색판건설을 방조하는 3개의 단추를 포함한다. 3-D Effects단추를 찰칵하여 Select Color대화칸을 펼친다. Background를 선택하여 Select Color대화칸을 펼친다. Tune Palette 단추를 찰칵하여 Tune Palette대화칸을 펼친다. Preview절의 Select Palette복합칸에서 미리볼 조색판을 선택한다.

OK를 찰칵하여 조색판에 대한 변경을 받아들인다. Cancel을 찰칵하여 조색판에 대한 변경을 보관하지 않고 대화칸을 닫는다.

4) Tune Palette대화칸

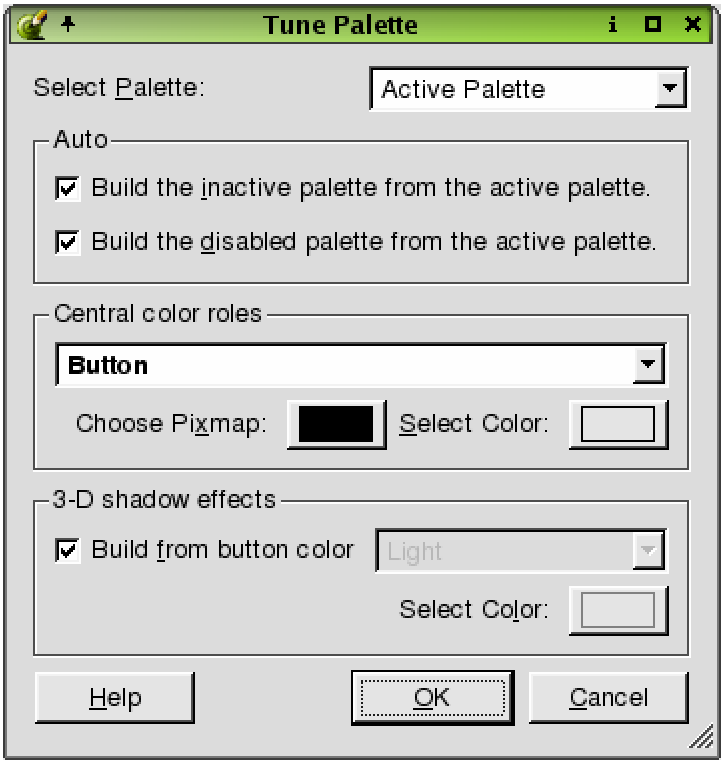


그림 1-134. Tune Palette

Tune Palette대화칸은 창문부품의 조색판용기능들을 선택하는데 리용된다.

Select Palette복합칸에서 능동, 비능동 혹은 금지하는 조색판의 선택들을 설정한다. Active Palette를 선택하면 대화칸이 조색판설계에 쓰이는 3개의 범주를 표시한다. 그 범주들은 Auto, Central Color Roles, 그리고 3-D Shadow Effects이다. Inactive Palette 혹은 Disabled Palette를 선택하면 모든 범주가 Auto를 제외하고 금지로 된다. Auto절의 검사칸들을 선택하여 능동조색판으로부터 능동 혹은 허용하지 않는 조색판들을 건설한다. 능동조색판에 대하여 Central color roles복합칸에서 조색판의 색역할을 선택한다. Choose Pixmap단추를 찰칵하여 Choose a Pixmap대화칸을 펼친다. Select Color단추를 찰칵하여 Select Color대화칸을 펼친다. 3-D shadow effects절의 Build from button color검사칸을 설정하여 단추색으로부터 3-D효과색을 계산할수 있게 한다. 검사칸을 해제하여 Choose 3-D effect color role복합칸이 가능하게 한다. 복합칸에서 3-D효과를 위한 색역할을 선택한다. Select Color단추를 찰칵하여 Select Color대화칸을 펼친다.

OK를 찰칵하여 조색판에 대한 변경을 받아들인다. Cancel을 찰칵하여 조색판에 대한 변경을 보관하지 않고 대화칸을 닫는다.

5) Select Font대화칸

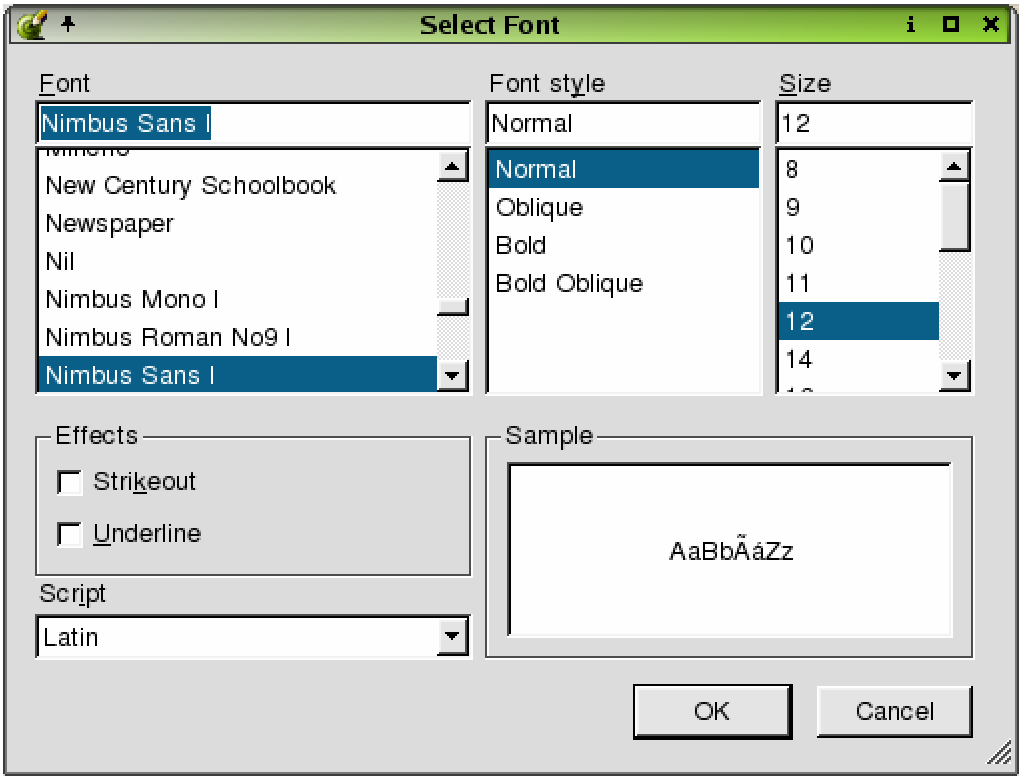


그림 1-135. Select Font

Select Font대화칸은 속성편집기의 서체속성열의 단추를 찰칵하여 펼친다. 이 대화칸을 리용하여 서체크기와 형식을 변경한다.

Font목록칸에서 서체형을 선택한다. 현재 선택된 형이 Font목록칸우의 행편집칸에 나타난다. Font Style목록칸에서 서체의 형식을 선택한다. 목록칸에서 유효한것은 선택하는 서체형으로 제한된다. 모든 서체가 유효한 모든 형식을 가지는것은 아니다. 선택된 형식은 Font Style목록칸우의 행편집칸에 나타난다. Size목록칸에서 서체의 크기를 선택한다. 현재 선택된 크기는 Size행편집칸우의 행편집칸에 나타난다. Effects절의 검사칸들을 선택하여 선택된 서체에 대한 Strikeout 혹은 Underline효과를 창조한다. Script를 선택하고 쓰기형식을 설정한다. Sample목록칸에 자기의 서체선택과 형식을 표시한다.

OK를 찰칵하여 서체에 대한 변경을 받아들인다. Cancel을 찰칵하여 서체에 대한 변경을 보관하지 않고 대화칸을 닫는다.

10. 통보칸

1) Save Project Settings대화칸

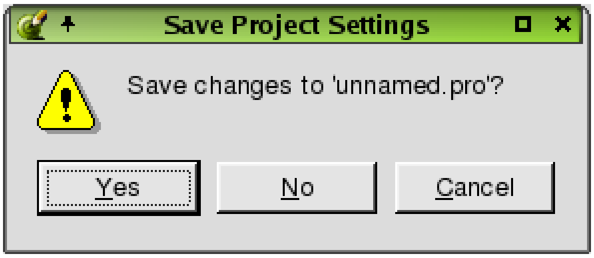


그림 1-136. Save Project Settings

Save Project Settings통보칸은 보관안된 변경이 있는 열려진 프로젝트에 대하여 File|Close 혹은 File|Exit를 눌러서 펼친다. 대화칸은 본문 'Save changes to your project.pro'?을 표시한다. Yes를 선택하여 변경을 보관한다. 프로젝트에 보관하지 못한 변경이 있는 폼이 있다면 Yes를 선택할 때 Save Form As대화칸이 펼쳐진다. No를 선택하여 변경을 보관하지 않고 프로젝트를 닫는다. Cancel을 찰칵하여 변경을 보관하지 않고 프로젝트를 닫는다.

2) Save Form대화칸

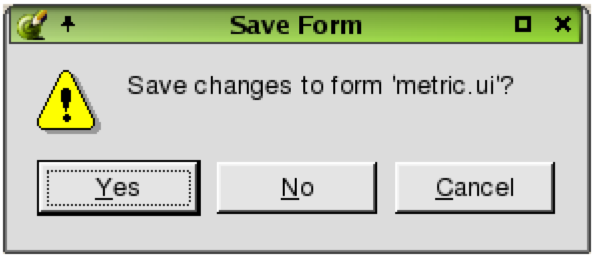


그림 1-137. Save Form

Save Form통보는 여러가지 방법으로 펼친다. 한가지 방법은 절대로 보관되지 않았거나 이전에 보관되였는데 변경이 있는 폼에 대하여 File|Close를 선택하는것이다. 대화칸은 또한 절대로 보관되지 않았거나 이전에 보관되였는데 변경이 있는 폼에 대하여 File|Exit를 선택하여 펼칠수 있다. 대화칸은 'Save Changes to the Form?'를 표시한다. Yes를 선택하여 폼을 보관한다. 폼이 이전에 보관되지 않았다면 Save Form As 대화칸이 열린다. No를 선택하여 변경을 보관하지 않고 폼을 닫거나 이미 보관하지 않았는데 폼을 보관하지 않고 폼을 닫는다. Cancel을 찰칵하여 폼을 보관하지 않고 폼을 닫거나 완료한다.

11. Object Explorer대화칸

일부 대화칸은 Object Explorer창문에서 유효한 선택들을 눌러서 펼친다.

1) Edit Forward Declarations대화칸

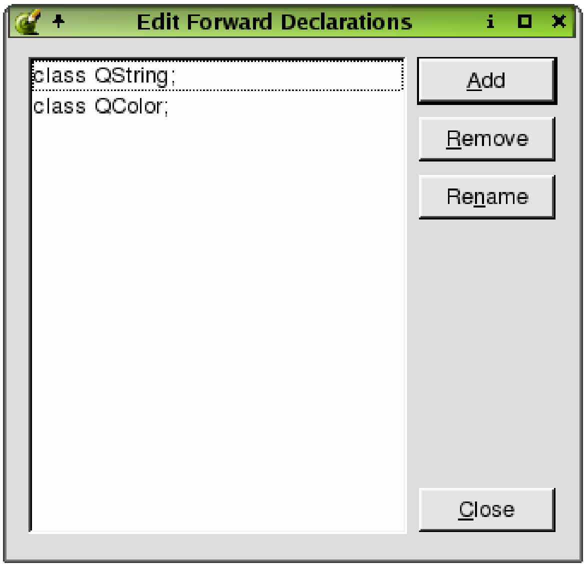


그림 1-138. Edit Forward Declarations

Object Explorer창문의 Source타브에서 Forward Declarations폴더를 오른쪽 단추로 선택하고 문맥차림표에서 Edit를 누르면 Edit Forward Declarations대화칸이 펼쳐진다. 이 대화칸을 리용하여 원천코드에 선언들을 추가, 편집, 혹은 삭제한다.

새로운 선언을 추가하려면 Add단추를 찰칵한다. 행편집칸이 나타나면 선언을 입력한다. 선언을 입력한 다음 Enter를 누른다. 목록칸에서 선언을 삭제하려면 그 선언을 선택하고 Remove를 찰칵한다. 현존선언의 이름을 변경하려면 그 선언을 선택하고 Rename을 누른다. 유표가 행편집칸에 나타나면 이름을 변경할수 있다.

Close를 선택하여 Edit Forward Declarations 대화칸을 닫는다.

2) Edit Includes (in Declaration)대화칸



그림 1-139. Edit Includes (in Declaration)

Object Explorer창문의 Source타브에서 Includes (in Declaration)홀더를 오른

쪽 단추로 선택하고 문맥차림표에서 Edit를 눌러서 Edit Includes (in Declarations) 대화칸을 펼친다. 이 대화칸을 리용하여 원천코드에 머리부를 추가, 편집, 혹은 삭제할 수 있다.

새 머리부를 추가하려면 Add단추를 찰칵한다. 행편집칸이 나타나면 머리부를 입력한다. 머리부를 입력한 다음 Enter를 누른다. 목록칸에서 머리부를 삭제하려면 머리부를 선택하고 Remove를 누른다. 현존머리부의 이름을 변경하려면 머리부를 선택하고 Rename을 찰칵한다. 유표가 행편집칸에 나타나면 이름을 변경할수 있다.

Close을 선택하여 Edit Include (in Declaration)대화칸을 닫는다.

3) Edit Includes (in Implementation)대화칸

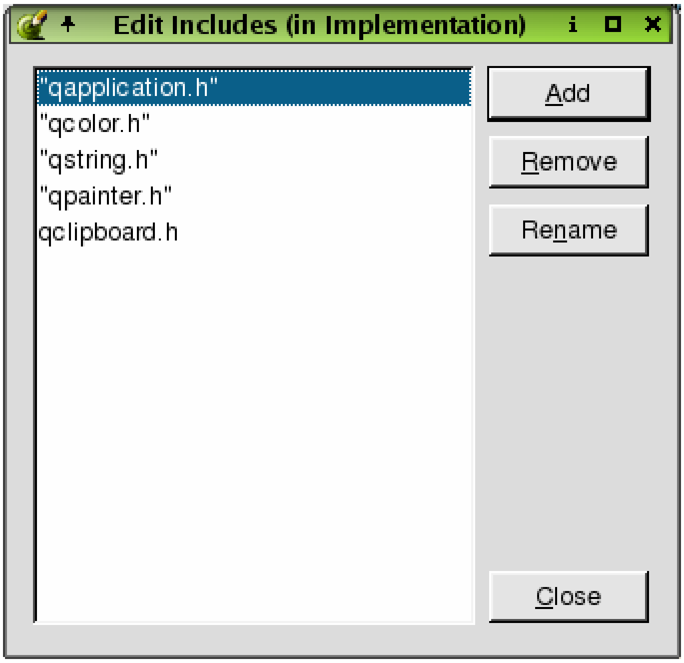


그림 1-140. Edit Includes (in Implementation)

Object Explorer창문의 Source타브에서 Includes (in Implementation)홀더를 오른쪽 단추로 선택하고 문맥차림표에서 Edit를 눌러서 Edit Includes (in Implementation)대화칸을 펼친다. 이 대화칸을 리용하여 원천코드에 머리부를 추가, 편집, 혹은 삭제할수 있다.

새 머리부를 추가하려면 Add단추를 찰칵한다. 행편집칸이 나타나면 머리부를 입력한다. 머리부를 입력한 다음 Enter를 누른다. 목록칸에서 머리부를 삭제하려면 머리부를 선택하고 Remove를 누른다. 현존머리부의 이름을 변경하려면 머리부를 선택하고 Rename을 찰칵한다. 유표가 행편집칸에 나타나면 이름을 변경할수 있다.

Close을 선택하여 Edit Include (in Implementation) 대화칸을 닫는다..

4) Edit Class Variables대화칸

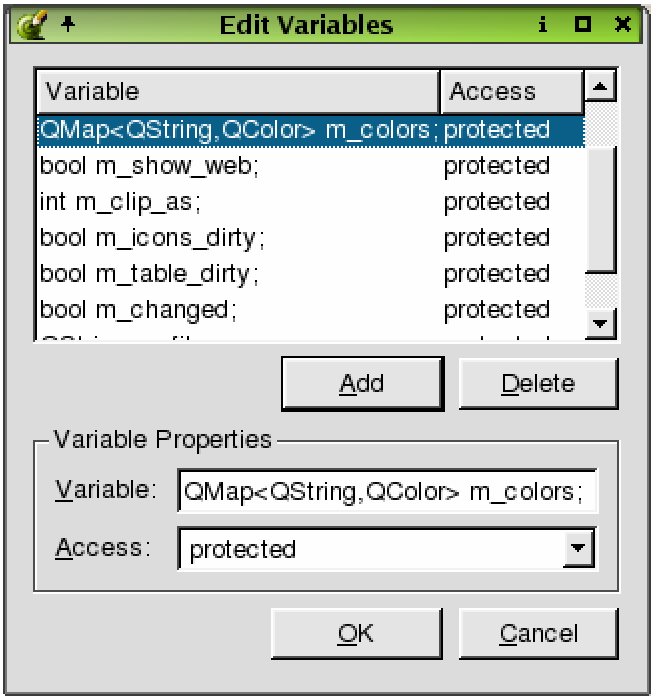


그림 1-141. Edit Class Variables

Object Explorer창문의 Source타브에서 Class Variables홀더를 오른쪽 단추로 선택하고 문맥차림표에서 Edit를 눌러서 Edit Class Variables대화칸을 펼친다. 이 대화칸을 리용하여 원천코드에 클라스변수들을 추가, 편집, 혹은 삭제할수 있다.

새 변수를 추가하려면 Add단추를 찰칵한다. 행편집칸이 나타나면 변수를 입력한다. 변수를 입력한 다음 Enter를 누른다. 목록칸에서 변수를 삭제하려면 변수를 선택하고 Remove를 누른다. 현존변수의 이름을 변경하려면 변수를 선택하고 Rename을 찰칵한다. 유표가 행편집칸에 나타나면 이름을 변경할수 있다.

Close를 선택하여 Edit Class Variables대화칸을 닫는다.

제13절. 위자드

Qt Designer에서 일부 도구띠들과 차림표선택, 형판들은 위자드를 기동하여 특정한 과제들을 한걸음씩 처리한다. 이 절에서는 매개의 Qt Designer위자드에 대하여 설명한다.

1. Main Window위자드

Main Window위자드는 New File대화칸의 Main Window폼형판을 선택하여 기동한다. 이 위자드는 작용과 차림표선택, 도구띠를 가지는 기본창문을 창조하는데 리용한다.

1) 유효한 차림표와 도구띠

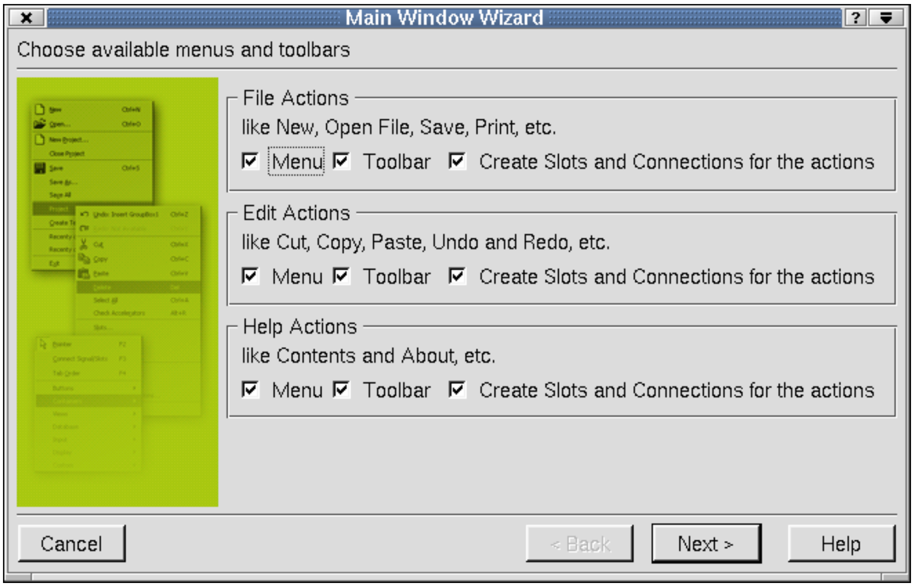


그림 1-142. Choose available menus and toolbars위자드페지

Choose available menus and toolbars위자드페지가 처음으로 나타난다. 이것은 기정작용들인 File작용과 Edit작용, Help작용을 제시한다. 매개 범주에 대하여 Qt Designer가 련관된 작용들을 위한 차림표항목들과 도구띠단추, 신호-처리부련결을 창조하도록 선택할수 있다. 후에 작용, 차림표항목, 도구띠단추, 련결을 추가하거나 삭제할수 있다. 검사칸들을 설정하거나 해제하여 자기가 좋아하는것을 선택할수 있다.

Next를 찰칵하여 다음의 위자드페지로 이동한다.

2) Setup도구띠위자드

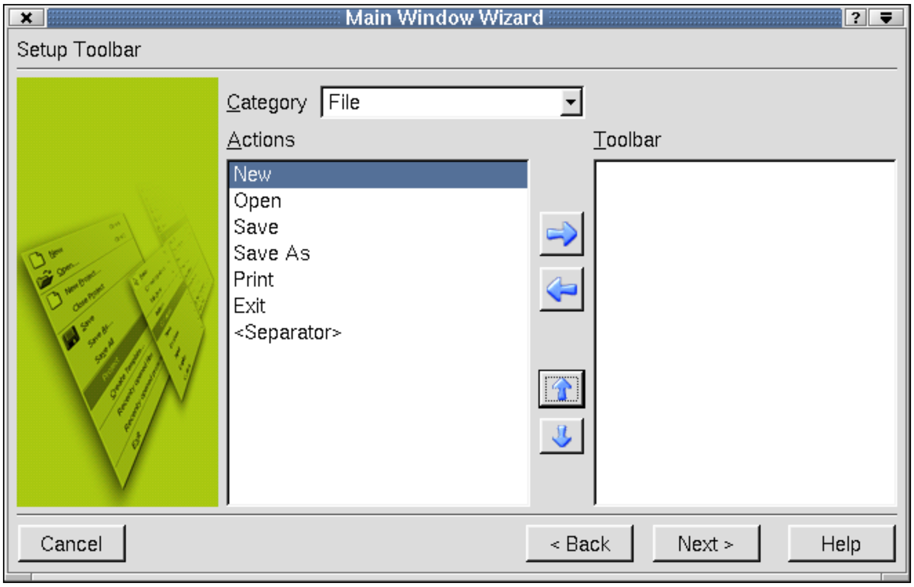


그림 1-143. Setup Toolbar위자드페지

Setup Toolbar위자드페지는 기정작용범주로부터 선택한 작용들을 가지는 도구띠를 만드는데 쓰인다. Category복합칸을 누르고 작업하려고 하는 작용들의 모임을 선택한다. Actions목록칸은 현재 범주에 쓸수 있는 작용들을 표시한다. Toolbar목록칸은 창조하려는 도구띠단추들을 표시한다. 푸른색의 왼쪽과 오른쪽 화살표를 선택하여 작용들을 Toolbar목록칸의 안이나 밖으로 이동한다. 푸른색의 올리 및 내리화살표를 눌러서 Toolbar목록칸의 우나 아래로 작용들을 이동한다. Actions목록칸의 <Separator>는 보통 필요할 때 Toolbar목록칸으로 이동될수 있으며 완료된 도구띠에 분리선이 나타나게 한다.

Choose available menus and toolbars위자드페지로 돌아오려면 Back를 찰칵한다. Finish를 선택하여 기본창문을 구성하고 위자드를 완료한다. 위자드의 임의의 페지에서 Cancel을 찰칵하여 변경없이 위자드를 닫는다.

2. Data Table위자드

Data Table위자드는 자료표창문부품을 선택하고 그것을 폼에 배치하면 자동적으로 기동한다. 자료표창문부품은 자료기지자료의 표보기를 창조하는데 리용된다.

1) 자료기지와 표의 선택(Choose the Database and Table)

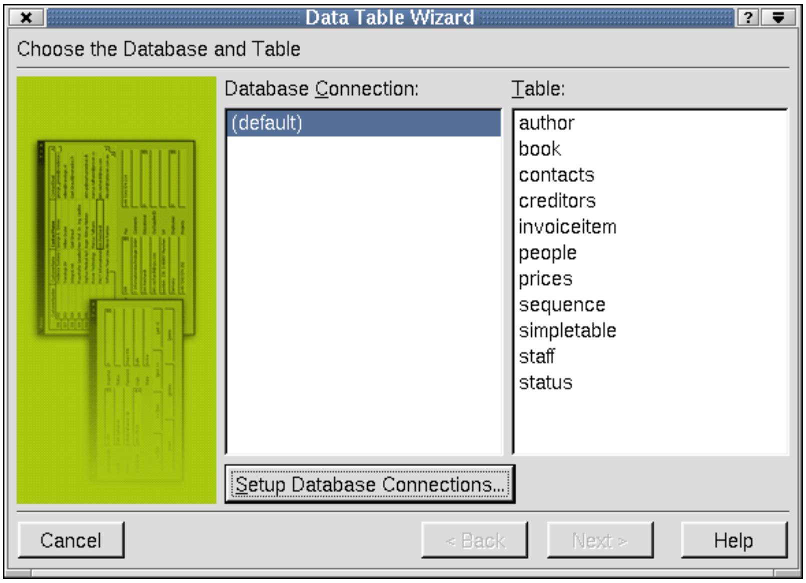


그림 1-144. Choose the Database and Table위자드페지

Choose the Database and Table위자드페지가 처음으로 나타난다. 유효한 자료기지들이 Database Connection목록칸에 표시된다. 그것을 눌러서 련결을 선택한다. 목록칸에 렬거된 련결이 없으면 Setup Database Connections를 선택하여 Edit Database Connections대화칸을 펼친다. Table목록칸에는 선택된 자료기지련결에서 유효한 표와 보기들을 모두 표시한다. 그것을 눌러서 표나 보기를 선택한다.

Next를 눌러서 다음의 위자드페지로 이동한다.

2) 현시된 마당(Displayed Fields)

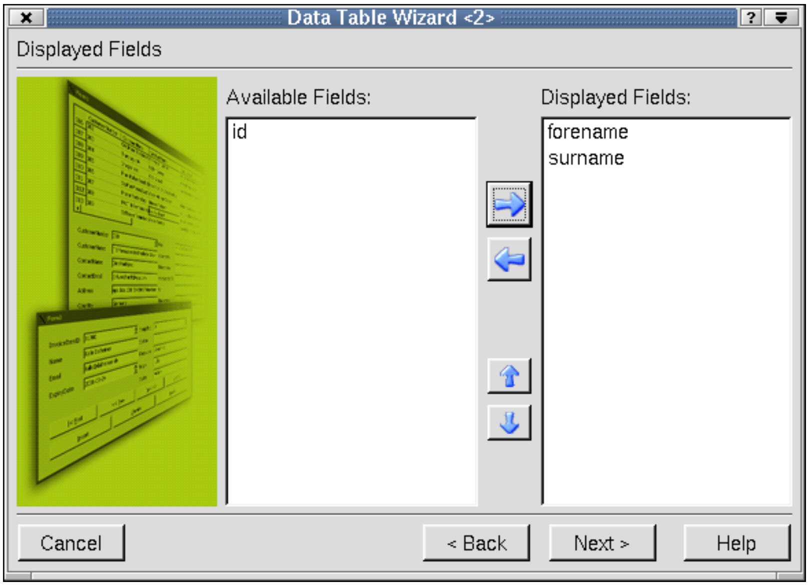


그림 1-145. Displayed Fields위자드페지

Displayed Fields위자드페지는 표에 표시될 마당들을 선택하는데 리용된다. 기정으로 표나 보기의 주열쇠를 제외한 매개 마당은 처음에 Displayed Fields목록에 배치된다. 푸른색의 왼쪽 및 오른쪽 화살단추를 찰칵하여 마당들을 Available Fields목록칸으로부터 Displayed Fields목록칸의 안이나 밖으로 이동한다. 푸른색의 우 및 아래화살단추들을 눌러서 Displayed Fields목록칸안의 우아래로 마당들을 이동한다. 마당들이 Displayed Fields목록칸에 나타나는 순서는 그것들이 Data Table에서 제일왼쪽란의 제일 꼭대기 마당에서부터 표시되는 순서이다.

Next를 찰칵하여 다음의 위자드페지로 이동한다. Choose the Database and Table 위자드페지로 돌아가려면 Back를 찰칵한다.

3) 표속성(Table Properties)

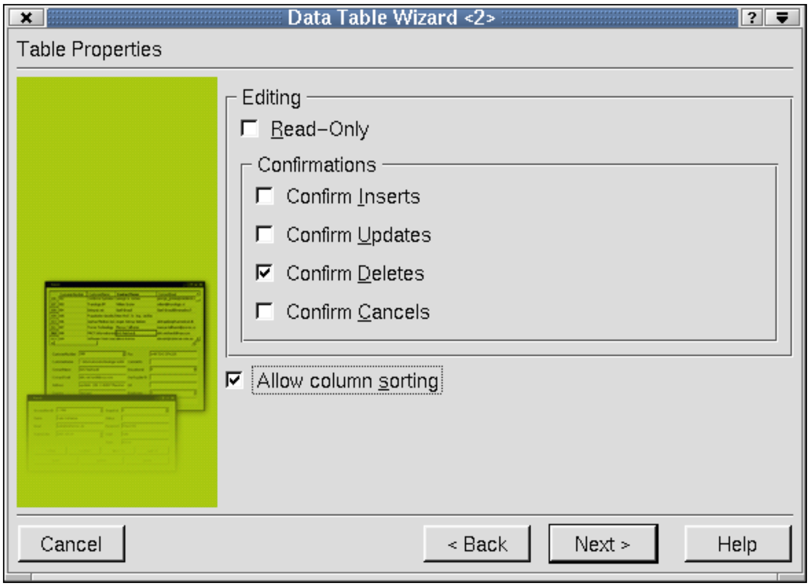


그림 1-146. Table Properties위자드페지

Table Properties위자드페지는 Data Table의 초기편집선택들을 설정하는데 쓰인다. Read-Only검사칸을 설정하여 레코드의 편집과 삭제, 추가를 방지한다. Confirmations절의 검사칸들을 설정하여 사용자가 변경을 확인하게 한다. 기정으로 사용자들은 삭제를 확인해야 한다. Allow column sorting을 눌러서 사용자가 렬머리부(마당이름을 표시한다)를 선택하여 자료를 정렬하게 한다.

Next를 찰칵하여 다음의 위자드페지로 이동한다. Displayed Fields위자드페지로 돌아가려면 Back를 찰칵한다.

4) SQL

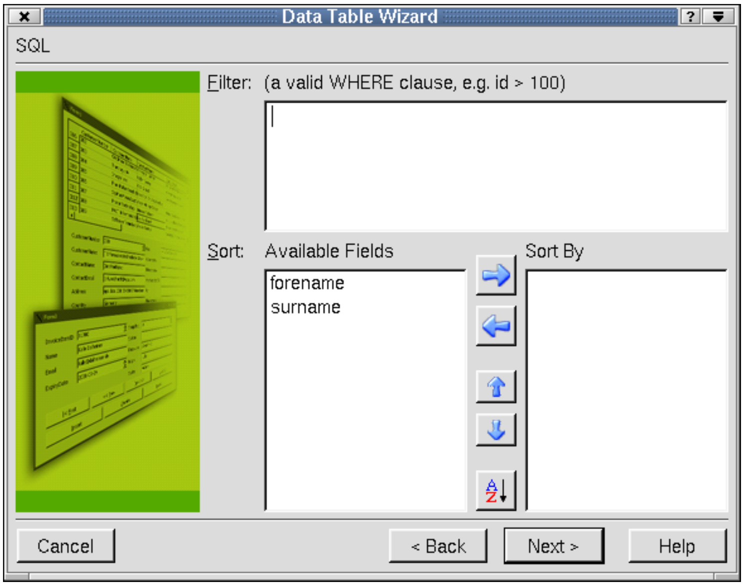


그림 1-147. SQL위자드페지

SQL위자드페지는 려과기들을 적용하여 표안의 자료를 정렬한다. Filter행편집칸을 선택하고 WHERE열쇠단어없이 유효SQL **WHERE**절을 입력한다. 려과기는 표에 표시되는 자료에 적용한다.

표안의 유효마당들을 정렬하려면 푸른색의 왼쪽 및 오른쪽 화살단추들을 선택하여 Available Fields목록칸으로부터 Sort By목록칸의 안이나 밖으로 마당들을 이동한다. 푸른색의 올리 및 내리화살단추들을 선택하여 Sort By목록칸안에서 우아래로 마당들을 이동한다. A-Z단추들을 선택하여 Sort By목록칸의 선택된 마당들의 정렬순서를 커지는 순서로, 혹은 작아지는 순서로 변경한다.

Next를 찰칵하여 다음의 위자드페지로 이동한다. Table Properties위자드페지로 돌아오려면 Back를 찰칵한다.

5) 완료(Finish)

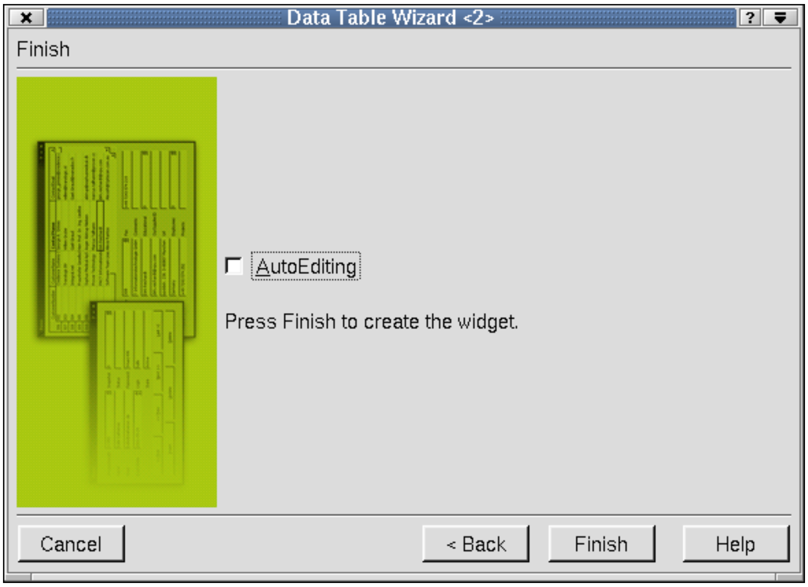


그림 1-148. Finish위자드페지

Finish위자드페지는 자동편집기능을 선택하고 위자드를 완료하는데 쓰인다. 사용자가 다른 레코드로 항행할 때 사용자편집 레를 들면 삽입과 갱신이 자동적으로 적용되게 하려면 AutoEditing검사칸을 설정한다. AutoEditing이 해제되면 사용자들은 Enter을 눌러서 다른 레코드로 이동하기전에 자기 편집을 확인해야 하며 그렇지 않으면 자기가 편집한 내용을 잃는다.

Finish를 선택하여 위자드에서 선택한 기능들을 모두 가진 자료기지창문부품을 창조한다. SQL위자드페지로 돌아가려면 Back를 찰칵한다. 변경없이 위자드를 완료하려면 임의의 위자드페지에서 Cancel을 찰칵한다.

3. Data Browser위자드

Data Browser위자드는 DataBrowser창문부품을 선택하고 그것을 폼우에 배치함으로써 자동적으로 호출된다. DataBrowser창문부품은 자료기지자료의 폼보기를 창조하는데 리용된다.

1) 자료기지와 표의 선택(Choose the Database and Table)

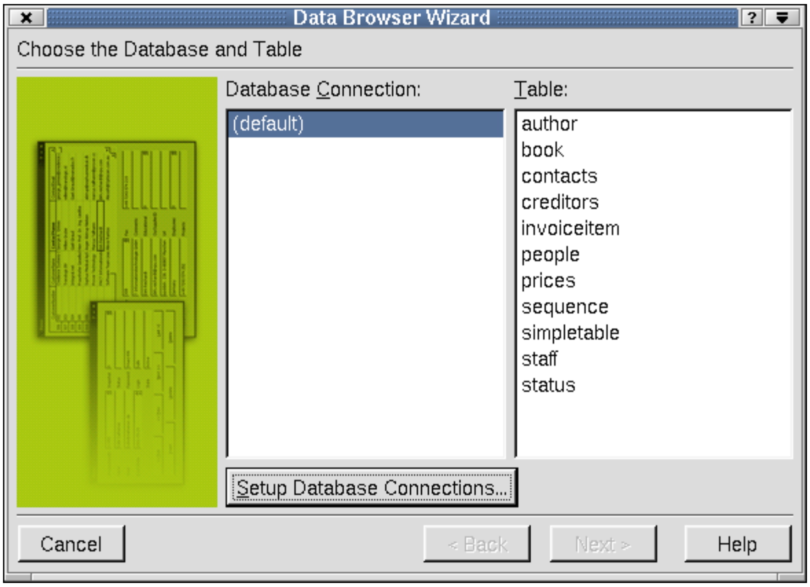


그림 1-149. Choose the Database and Table위자드페지

Choose the Database and Table위자드페지가 처음에 나타난다. 유효한 자료기지들이 Database Connection목록칸에 표시된다. 련결을 눌러서 선택한다. 목록칸에 어떤 련결도 표시되여있지 않으면 Setup Database Connections을 눌러서 Edit Database Connections대화칸을 연다. Table목록칸은 선택된 자료기지련결에서 유효한 모든 표와 보기들을 표시한다. 표나 보기를 하나 눌러서 선택한다.

Next를 찰칵하여 다음의 위자드페지로 이동한다.

2) 현시된 마당들(Displayed Fields)

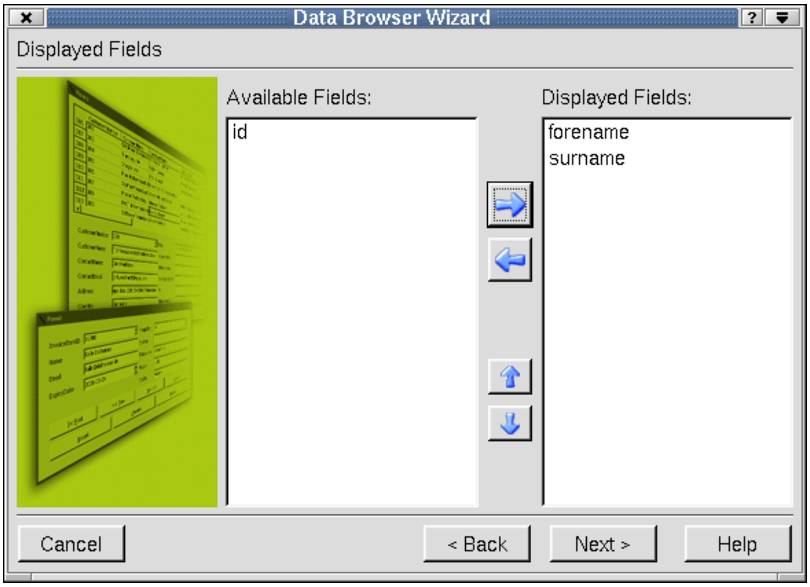


그림 1-150. Displayed Fields위자드페지

Displayed Fields위자드페지는 표에 표시하려는 마당들을 선택하는데 사용된다. 푸른색의 왼쪽 및 오른쪽 화살단추들을 눌러서 Available Fields목록칸으로부터 Displayed Fields목록칸의 안이나 밖으로 마당들을 이동한다. 푸른색의 올리 및 내리화살단추들을 눌러서 Displayed Fields목록칸안에서 우아래로 마당들을 이동한다.

Next를 찰칵하여 다음의 위자드페지로 이동한다. Choose the Database and Table 위자드페지로 돌아가려면 Back를 찰칵한다.

3) 항행과 편집(Navigation and Editing)

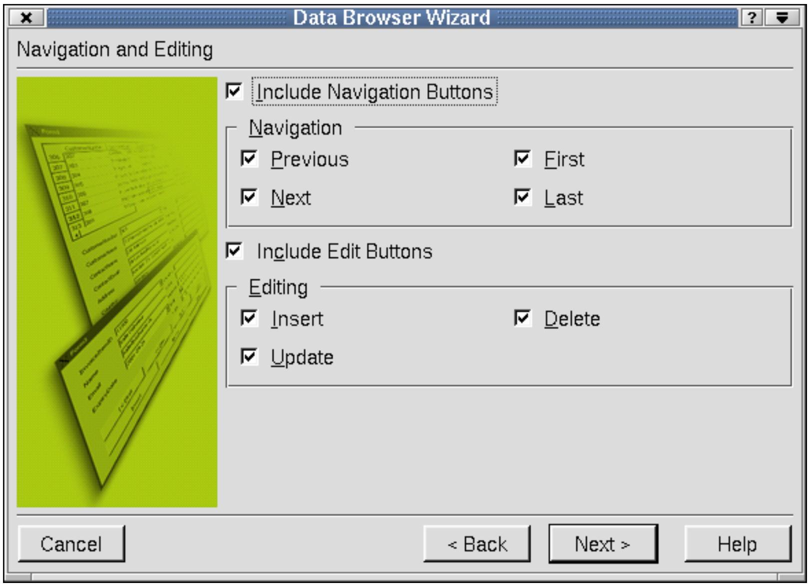


그림 1-151. Navigation and Editing위자드페지

Navigation and Editing위자드페지는 항행과 편집단추들을 창조하는데 리용된다.

Include Navigation Buttons검사칸을 설정하여 항행단추들을 포함한다. Navigation절에서 Previous를 선택하여 폼우에 Previous단추를 표시한다. 이 선택은 표안의 이전의 레코드로 항행하게 한다. Next를 찰칵하여 폼우에 next단추를 표시한다. 이 단추는 표안의 다음의 레코드로 항행하게 한다. First를 선택하여 폼우에 First단추를 표시한다. 이 선택은 표안의 첫 레코드로 항행하게 한다. Last를 선택하여 폼우에 Last단추를 표시한다. 이 단추는 표안의 마지막 레코드로 항행하게 한다.

Include Edit Buttons검사칸을 설정하여 편집단추들을 포함한다. Editing절에서 Insert검사칸을 설정하여 새 레코드를 추가하기 위한 Insert단추를 창조한다. Update검사칸을 설정하여 현존 레코드들을 갱신하기 위한 Update단추를 창조한다. Delete검사칸을 설정하여 레코드들을 삭제하기 위한 Delete단추를 창조한다.

항행단추들과 Update 및 Delete단추는 코드를 추가하지 않아도 동작한다. 대부분의 자료기지들은 유일건을 가지는 새 레코드창조기능이 없이 설계되므로 Insert단추는

동작하지 않는다. 이것은 QDataBrowser::beforeInsert()신호에 련결된 처리부를 만들어 간단히 해결할수 있다.

Next를 찰칵하여 다음의 위자드페지로 이동한다. Displayed Fields위자드페지로 돌아가려면 Back를 찰칵한다.

4) SQL

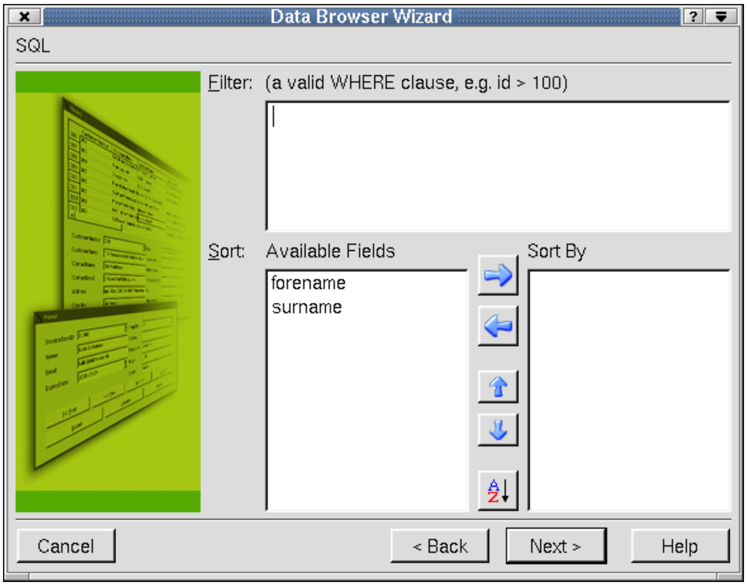


그림 1-152. SQL위자드페지

SQL위자드페지는 려과기들을 적용하여 표안의 자료를 정렬한다. Filter행편집칸을 선택하고 WHERE열쇠단어없이 유효SQL WHERE절을 입력한다. 려과기는 표에 표시되는 자료에 적용한다.

표안의 유효마당들을 정렬하려면 푸른색 왼쪽 및 오른쪽 화살단추들을 선택하여 Available Fields목록칸으로부터 Sort By목록칸의 안이나 밖으로 마당들을 이동한다. 푸른색 올리 및 내리화살단추들을 선택하여 Sort By목록칸안에서 우아래로 마당들을 이동한다. A-Z단추들을 선택하여 Sort By목록칸의 선택된 마당들의 정렬순서를 커지는 순서로, 혹은 작아지는 순서로 변경한다.

Next를 찰칵하여 다음의 위자드페지로 이동한다. Navigation and Editing위자드페지로 돌아가려면 Back를 찰칵한다.

5) 배치(Layout)

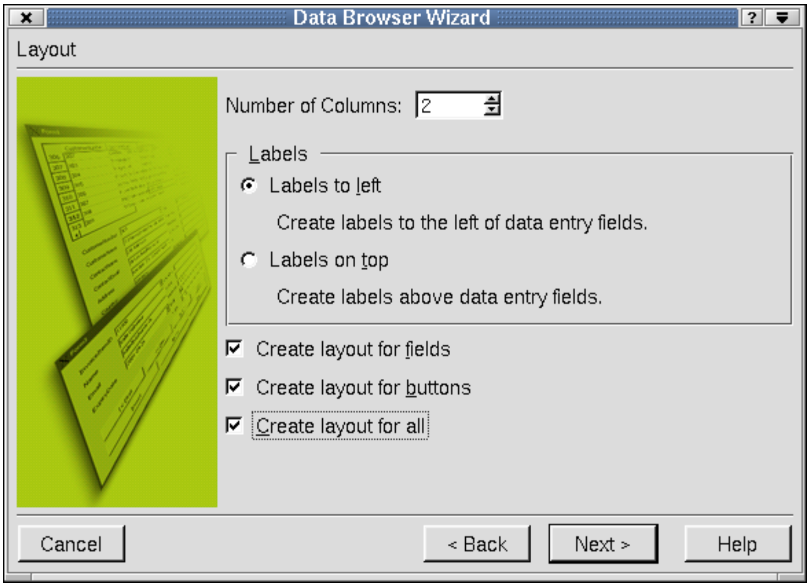


그림 1-153. Layout위자드페지

Layout위자드페지는 자료기지열람기의 배치를 설계하는데 리용된다. 폼이 사용할 렬수를 선택하려면 Number of Columns스핀칸을 선택한다. 자료항목마당들의 왼쪽에 표식자들이 나타나게 하려면 Labels to left라지오단추를 찰칵한다. 자료항목마당들의 우에 표식자들이 나타나게 하려면 Labels on top라지오단추를 찰칵한다.

Create layout for fields검사칸을 선택하여 칸배치안쪽의 모든 창문부품들을 배렬한다. Create layout for buttons검사칸을 설정하여 칸배치안쪽의 모든 단추들을 배렬한다. Create layout for all를 선택하여 전체 창문부품을 위한 칸배치를 창조한다.

항상 배치를 해체하고 그것들을 후에 재시행할수 있다.

Next를 찰칵하여 다음의 위자드페지로 이동한다. SQL위자드페지로 돌아가려면 Back를 찰칵한다.

6) 완료(Finish)

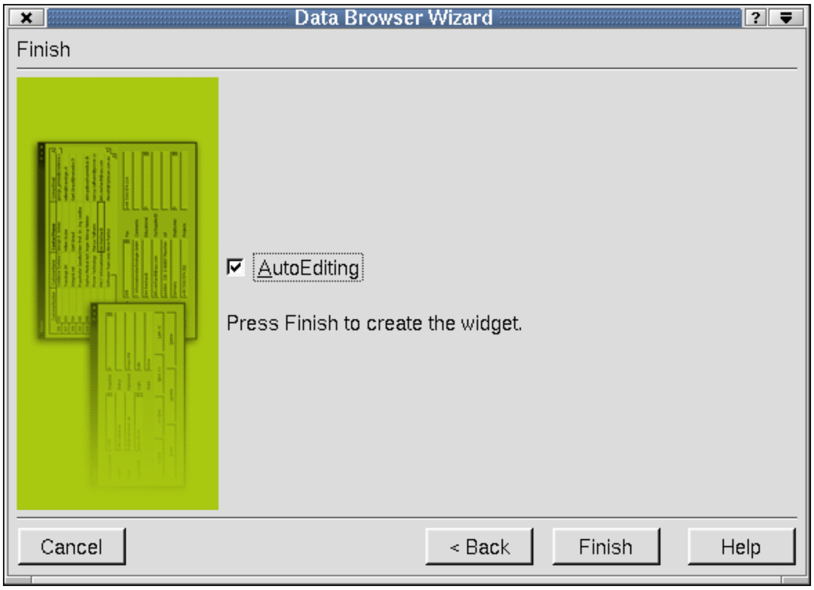


그림 1-154. Finish위자드페지

Finish위자드페지는 자동편집기능을 선택하고 위자드를 완료하는데 쓰인다. 사용자가 다른 레코드로 항행할 때 사용자편집 레를 들면 삽입과 갱신이 자동적으로 적용되게 하려면 AutoEditing검사칸을 설정한다. AutoEditing이 해제되면 사용자들은 Enter을 눌러서 다른 레코드로 이동하기전에 자기 편집을 확인해야 하며 그렇지 않으면 자기가 편집한 내용을 잃는다. 이 속성은 후에 필요할 때 변경할수 있다.

Finish를 선택하여 위자드에서 선택한 기능들을 모두 가진 자료기지창문부품을 창조한다. Layout위자드페지로 돌아가려면 Back를 찰칵한다. 임의의 위자드페지에서 Cancel을 찰칵하여 변경없이 위자드를 완료할수 있다.

4. Data View위자드

Data View위자드는 자료보기창문부품을 선택하고 폼우에 그것을 배치하여 자동적으로 호출될수 있다. Dataview창문부품은 자료기지자료의 읽기전용폼보기를 창조하는데 쓰인다.

1) 자료기지와 표의 선택(Choose the Database and Table)

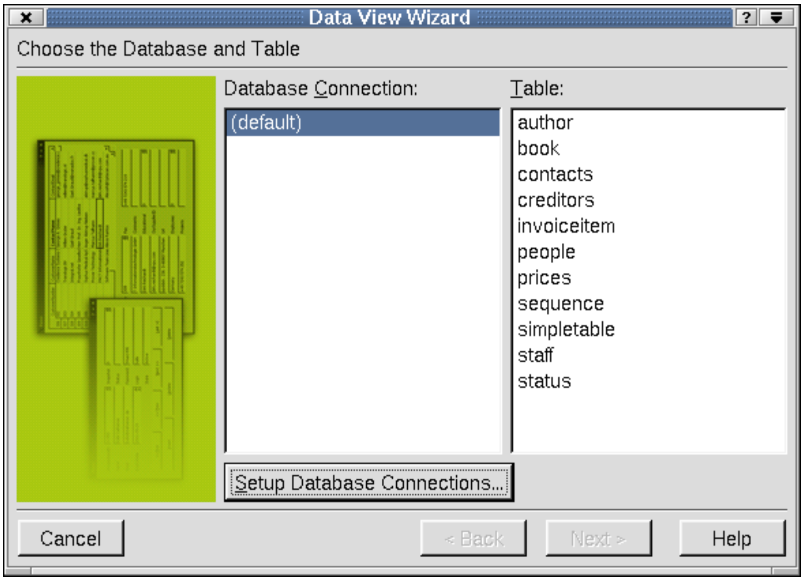


그림 1-155. Choose the Database and Table위자드페지

Choose the Database and Table위자드페지가 처음으로 나타난다. 유효한 자료기지들이 Database Connection목록칸에 표시된다. 그것을 눌러서 련결을 선택한다. 목록칸에 렬거된 련결이 없으면 Setup Database Connections를 선택하여 Edit Database Connections대화칸을 펼친다. Table목록칸에는 선택된 자료기지련결에서 유효한 표와 보기들을 모두 표시한다. 그것을 눌러서 표나 보기를 선택한다.

Next를 찰칵하여 다음의 위자드페지로 이동한다.

2) 현시된 마당(Displayed Fields)

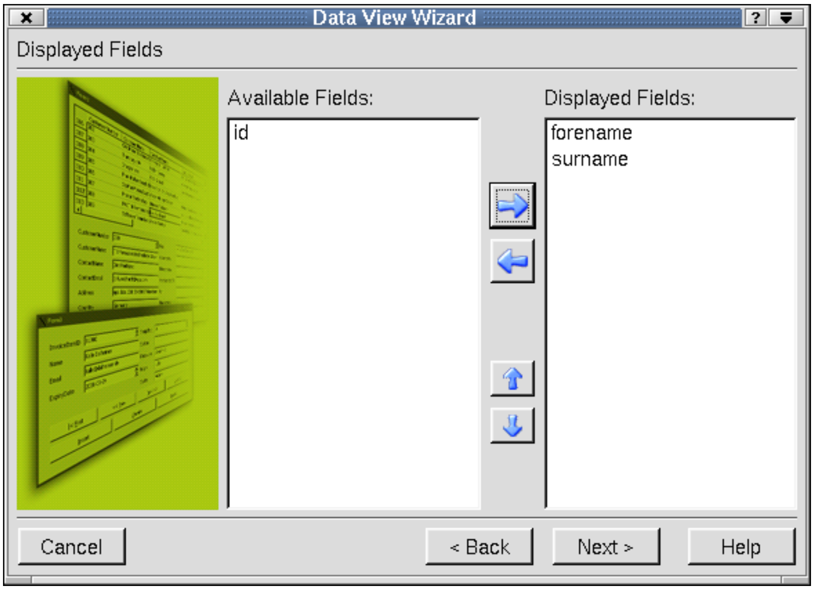


그림 1-156. Displayed Fields위자드페지

Displayed Fields위자드페지는 표에 표시될 마당들을 선택하는데 리용된다. 푸른색의 왼쪽 및 오른쪽 화살단추를 찰칵하여 마당들을 Available Fields목록칸으로부터 Displayed Fields목록칸의 안이나 밖으로 이동한다. 푸른색의 우 및 아래화살단추들을 눌러서 Displayed Fields목록칸안의 우아래로 마당들을 이동한다.

Next를 찰칵하여 다음의 위자드페지로 이동한다. Choose the Database and Table 위자드페지로 돌아가려면 Back를 찰칵한다.

3) 배치(Layout)

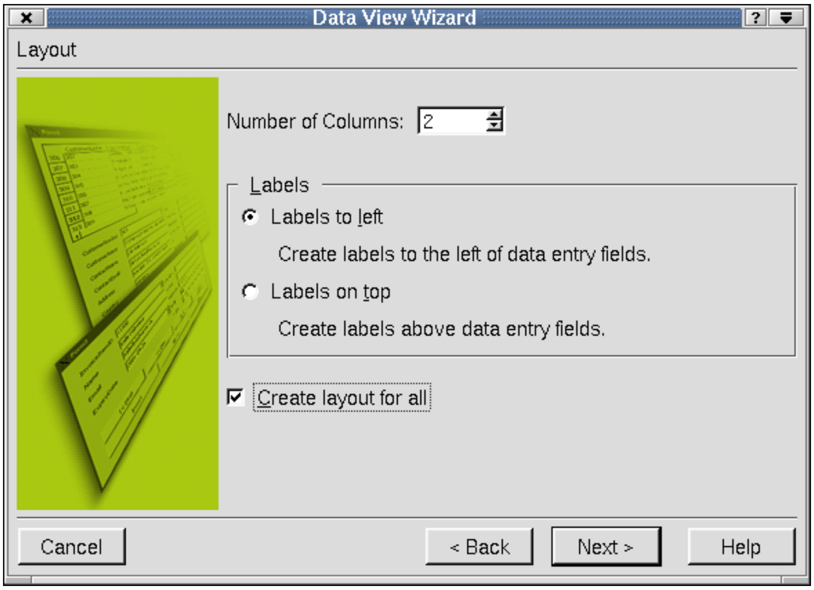


그림 1-157. Layout위자드페지

Layout위자드페지는 자료기지보기의 배치를 설계하는데 리용된다. 폼이 사용할 렬수를 선택하려면 Number of Columns스핀칸을 선택한다. 자료항목마당들의 왼쪽에 표식자들이 나타나게 하려면 Labels to left라지오단추를 찰칵한다. 자료항목마당들의 우에 표식자들이 나타나게 하려면 Labels on top라지오단추를 찰칵한다.

Next를 찰칵하여 다음의 위자드페지로 이동한다. Displayed Fields위자드페지로 돌아가려면 Back를 찰칵한다.

4) 완료(Finish)

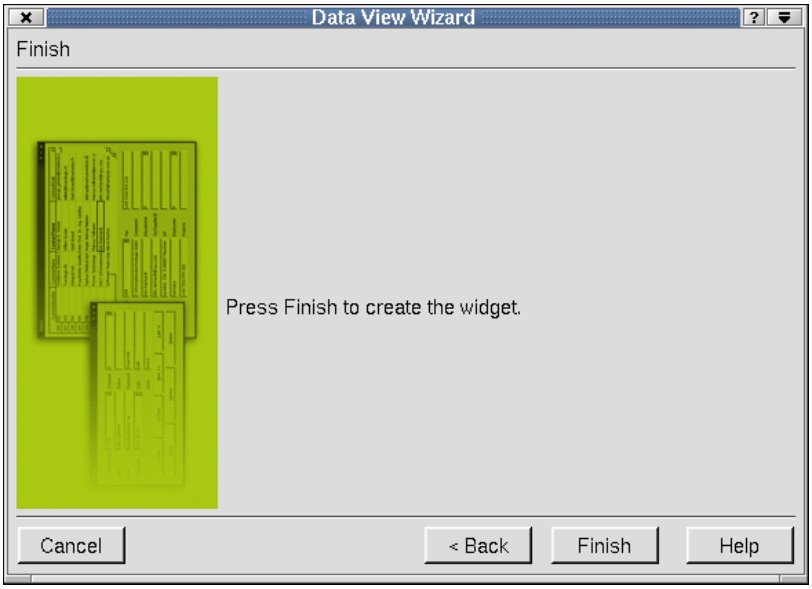


그림 1-158. Finish위자드페지

Finish위자드페지는 이전의 위자드페지들에서 요구하는 선택들을 모두 설정한 다음에 위자드를 창조하는데 쓰인다.

Finish를 찰칵하여 위자드에서 선택한 기능을 모두 가지는 자료열람기창문부품을 창조한다. Layout위자드페지로 돌아가려면 Back를 찰칵한다. 임의의 위자드페지에서 Cancel을 찰칵하여 변경을 보관하지 않고 위자드를 완료한다.

제14절. 창문

Qt Designer는 기동할 때 왼쪽에 3개의 창문을 펼친다. 이것들은 Project Overview창문, Object Explorer창문, 그리고 Property Editor/Signal Handlers창문이다. 또한 Qt Designer는 폼의 차림표용작용들과 차림표항목들을 창조하기 위한 Action Editor를 제공한다. 이 절에서는 매개창문들에 대하여 자세히 설명한다.

1. Project Overview창문

이 창문은 프로젝트와 련관된 파일들을 모두 렬거한다. Files목록에서 폼이나 파일을 한번 눌러서 그것을 펼친다. 폼과 파일들사이를 고속으로 절환하기 위하여 파일목록우의 행편집칸에 파일이름을 입력하면 Qt Designer는 증분탐색을 수행하여 일치하는 파일들이나 폼들을 표시한다.

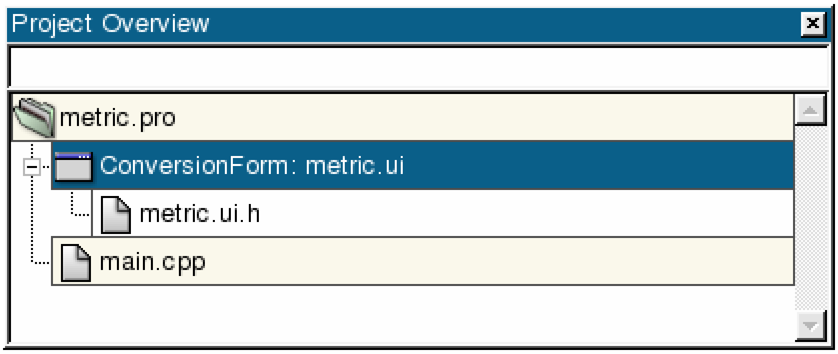


그림 1-159. Project Overview창문

파일(혹은 프로젝트)를 오른쪽 단추로 눌러서 선택용문맥차림표 실례로 Open form이나 Remove form from project를 얻는다.

2. Object Explorer창문

Object Explorer창문은 현재 폼의 창문부품과 처리부들을 렬거한다. 창문에는 두개의 타브 즉 Objects타브와 Members타브가 있다.

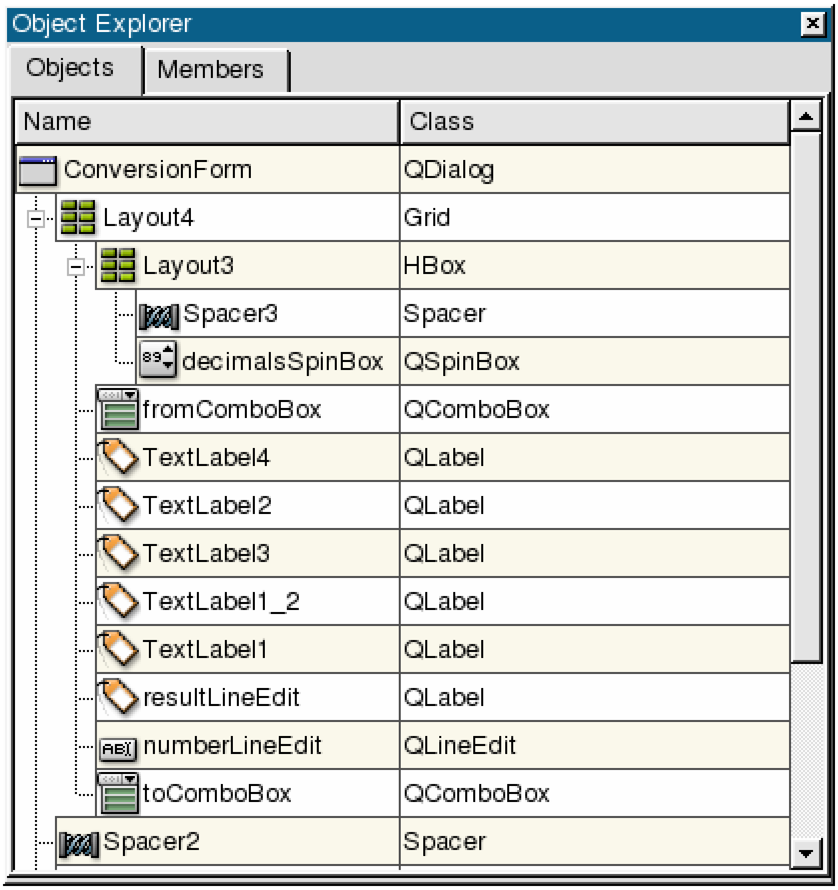


그림 1-160. Objects타브

1) Objects타브

Widgets타브를 선택하여 현재 폼의 창문부품들을 모두 표시한다. 창문부품들은 이름과 클라스별로 렬거된다. 목록안의 창문부품을 선택하여 대응하는 폼안에서 그것을 강조표시한다.

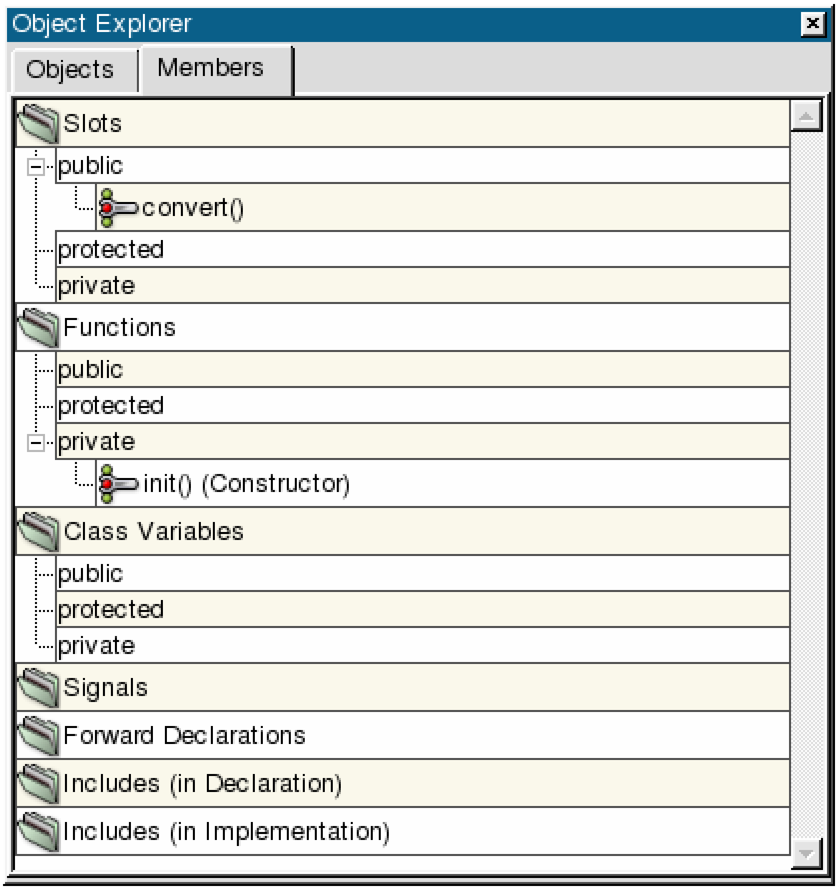


그림 1-161. Members타브

2) Members타브

Members타브를 선택하면 현재 폼의 처리부들, 앞방향선언들, 머리부들, 그리고 클라스변수들을 표시한다. Members타브는 나무보기를 사용하여 그 정보를 현시하다. +부호를 가지는 항목들은 +를 누르면 펼쳐지는 보조항목들을 가지고있다. 나무보기의 항목을 오른쪽 단추로 눌러서 문맥차림표를 펼친다.

처리부를 편집 혹은 추가하려면 Slots폴더를 오른쪽 단추로 찰칵하고 Edit를 선택하여 Edit Slots대화칸을 연다. Public, Protected 혹은 Private보조등록부들을 오른쪽 단추로 찰칵하고 New를 선택하여 Edit Slots대화칸을 펼친다. 목록안의 처리부를 오른쪽 단추로 눌러서 그 처리부용의 추가적인 선택이 있는 차림표를 펼친다. 새 처리부들을 추가하려면 차림표에서 New를 선택하여 Edit Slots대화칸을 펼친다. 선택된 처리부의 속성들을 변경하려면 Properties를 선택하여 Edit Slots대화칸을 연다. C++편집기를 열고 선택된 처리부의 실현으로 이행하려면 Goto Implementation을 선택한다. 선택된 처리부를 삭제하려면 Delete를 누른다. 신호를 처리부와 같은 방법으로 추가하거나

삭제할수 있다.

Forward Declarations와 Includes (in declaration), Class Variables, Includes (in implementation)를 오른쪽 단추로 눌러서 new 혹은 edit선택을 가지는 문맥차림표를 펼친다. New를 선택하여 선언, 변수 혹은 머리부를 입력하기 위한 행편집칸을 펼친다. Forward Declarations를 오른쪽 단추로 찰칵하고 Edit를 선택하여 Edit Forward Declarations대화칸을 연다. Includes (in declaration)를 오른쪽 단추로 찰칵하고 Edit를 선택하여 Edit Includes (in Declaration)대화칸을 연다. Class variables를 오른쪽 단추로 찰칵하고 Edit를 선택하여 Edit Class Variables대화칸을 연다. Includes (in Implementation)를 오른쪽 단추로 찰칵하고 Edit를 선택하여 Edit Includes (in Implementation)대화칸을 연다.

3. Property Editor/Signal Handlers창문

Property Editor/Signal Handlers창문을 선택하여 폼과 창문부품, 차림표들의 속성을 표시하고 변경한다. 이 창문에는 Properties타브와 Signal Handlers타브가 있다.

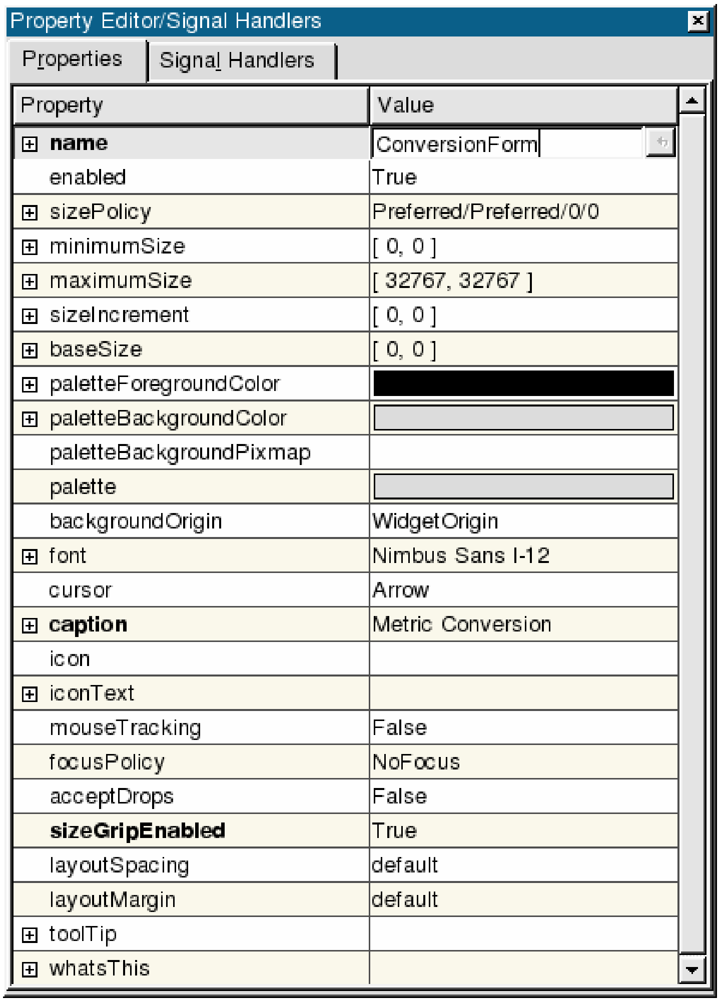


그림 1-162. Properties타브

1) Properties타브

Properties타브를 선택하여 선택된 창문부품의 외관과 동작을 변경한다. (차림표인 경우에는 차림표띠를 선택하여 Property Editor에 차림표항목의 속성들을 표시한다.) Property Editor에는 두개의 렬 즉 속성이름들을 렬거하는 Property렬과 값들을 렬거하는 Value렬이 있다. 렬의 머리부를 를 선택하여 속성이나 값들을 정렬한다. 일부 속성이름들은 왼쪽의 정방형안에 +기호를 가지고있다. 이것은 속성이름이 련관된 속성들의 모임에 대한 집합이름이라는것을 의미한다.

일부 속성들은 단일값을 가진다. 실례로 이름속성은 본문값을 가지고 폭속성(실례에서 minimumSize)은 수값을 가진다. 본문값을 변경하려면 현존 본문을 선택하고 새 본문을 입력한다. 수값을 변경하려면 그 값을 선택하고 새로운 수를 입력하거나 스핀단추를 누르고 현존 수값이 요구하는 값에 이를 때까지 증가 혹은 감소시킨다. 일부 속성들은 고정된 값목록을 가진다. 실례로 mouseTracking속성은 론리형으로서 True 혹은 False값을 가질수 있다. 유표속성도 역시 고정된 값목록을 가진다. 유표속성이나 mouseTracking속성을 선택하면 그 값이 내리펼침복합칸에 나타나는데 아래방향화살표를 눌러서 유효한 값들을 알아볼수 있다.

일부 속성은 값들의 복잡한 모임을 가진다. 실례로 서체속성이 있다. 서체속성을 선택하면 생략단추 (...)가 나타나는데 이 단추를 찰칵하면 Select Font대화칸이 펼쳐지고 여기서 서체설정을 변경할수 있다. 다른 속성들도 그 속성이 가지는 설정에 따라서 서로 다른 대화칸을 펼치는 생략단추들을 가지고있다. 실례로 text속성에 입력할 본문이 많다면 생략단추를 찰칵하여 여러행본문편집대화칸을 펼친다. 변경되는 속성들의 이름은 강조체로 표시된다. 속성을 변경하였는데 그 기정값으로 되돌려보내려고 하지 않으면 그 속성값을 선택하고 값의 오른쪽에 있는 붉은색의 X단추를 찰칵한다. 일부 속성들은 초기값 실례로 TextEdit1을 가지지만 기정값은 없다. 초기값을 가지지만 기정값이 없는 속성을 되살리면(붉은색의 X단추를 찰칵하여) 그 값은 속성 실례로 이름이 비는것을 허용하지 않으면 비게 된다.

속성편집기는 Undo와 Redo를 완전히 지원한다. (Ctrl+Z와 Ctrl+Y를 Edit차림표로부터 사용할수 있다.).

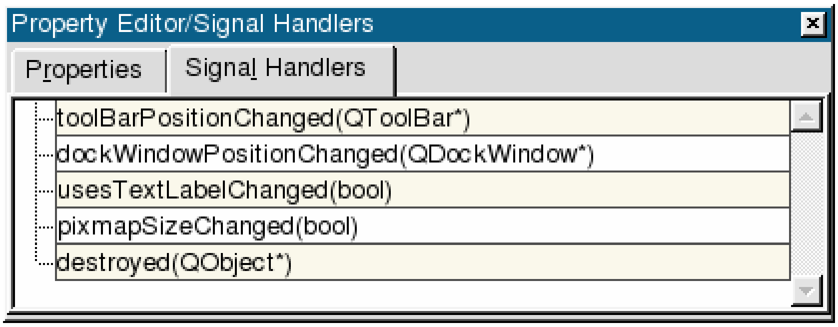


그림 1-163. Signal Handlers타브

2) Signal Handlers타브

Signal Handlers타브를 선택하여 창문부품들의 신호들사이의 련결과 폼의 전용처리부들을 표시하거나 창조한다.

4. Action Editor창문

Action Editor창문은 사용자대면부를 통하여 사용자가 초기화하는 작용들을 창조하는데 쓰인다. 실례로 파일의 보관이나 인쇄. Action Editor창문을 호출하려면 기본창문이 능동일 때 Window|Views|Action Editor를 선택한다.

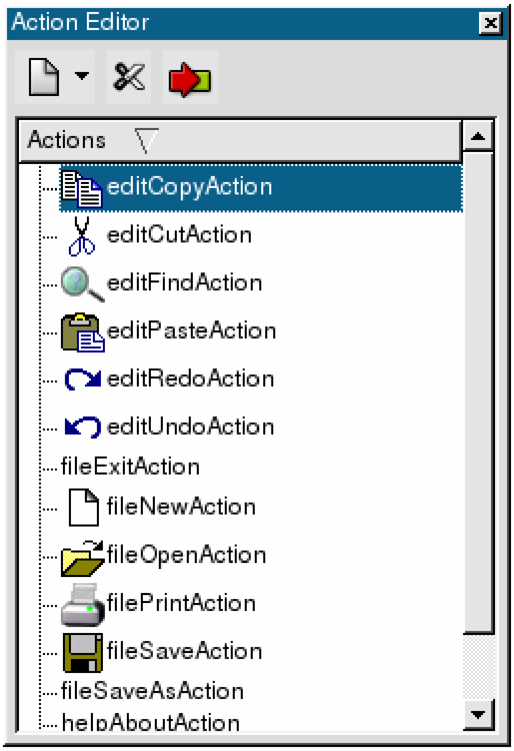


그림 1-164. Action Editor

Action Editor는 그 차림표띠에 3가지 기능 즉 창조, 삭제 및 련결기능을 가지고있다. 또한 현존작용들을 렬거하는 목록보기를 가지고있다.

새로운 작용을 창조하려면 차림표띠에서 Create a New Action그림기호우의 화살표를 선택한다. 새로운 작용, 새로운 작용그룹, 또는 새로운 내리펼침작용그룹을 선택하여 창조한다. 또한 Action Editor목록보기에서 현존 작용을 오른쪽 단추로 찰칵하고 튀여나오기차림표에서 창조하려는 작용형을 선택하여 작용들을 창조할수 있다.

목록보기에서 작용을 삭제하려면 목록에서 삭제하려는 작용을 선택하고 차림표띠에서 Delete Current Action를 선택한다. 또한 Action Editor목록보기에서 현존 작용을 오른쪽 단추로 찰칵하고 튀여나오기차림표로부터 Delete Action을 선택하여 작용들을 삭제할수 있다.

작용을 처리부와 련결하려면 목록보기우에서 작용을 선택하고 차림표띠에서 Connect그림기호를 눌러서 View and Edit Connections대화칸을 펼친다. 또한 Action Editor목록보기에서 현존작용을 오른쪽 단추로 선택하고 튀여나오기차림표의 Connect Action을 누름으로써 작용을 처리부와 련결할수 있다.

제15절. .ui파일형식

Qt Designer .ui파일에 폼을 보관한다. 이 파일들은 XML형식으로 폼요소들과 그 특성을 표시한다. 여기서는 XML형식에 대한 개괄을 주며 개발자들이 자체로 .ui구문해석기를 쓰는데 충분한 정보를 제공하여 .ui파일들을 프로그람적으로 읽고 수정할수 있게 하려고 한다.

.ui파일의 문법을 해석하는 한가지 방법은 Qt의 QDomDocument클라스를 사용하는것이다. 이것은 Qt Designer가 이 클라스를 사용하는 방법이다. 원천파일 uilib/qwidgetfactory.h과 uilib/qwidgetfactory.cpp을 참고하시오.

.ui파일의 문서형은 간단히 UI이므로 doctype꼬리표는 다음과 같다.

```
<!DOCTYPE UI>
```

뿌리요소는 UI로서 전체 내용을 둘러싼다.

```
<UI version="3.1" stdsetdef="1">
...
</UI>
```

UI항목안에서 다음과 같은 요소형의 하나이거나 없을수 있다.

actions - QMainWindow폼용작용들

author - 폼의 저자

class - 폼의 클라스이름

comment - 설명 실례로 저작권표기

connections - 신호-처리부련결

customwidgets - 사용자정의창문부품(낡은 형식)

exportmacro - Windows에 고유

forwards - 앞방향선언

functions - 함수선언

images - 매몰된 화상: 매몰된 화상들을 포함하는 .ui파일들에 대해서만. 화상들은 보통 개별적인 images 등록부에 보관된다.

includes - 머리부파일

layoutdefaults - 배치속성의 기정값

layoutfunctions - 배치속성의 동적기정값

menubar - QMainWindow 폼들의 차림표띠

pixmapfunction - 매몰형픽스매프나 외부 픽스매프들이 사용되지 않으면 픽스매프들을 얻는데 사용하는 함수이름

pixmapinproject - 픽스매프가 .pro파일에 의하여 조종된다는것을 가리키는 요소

signals - 신호선언

slots - 처리부선언

tabstops - 폼의 타브순서

toolbars - QMainWindow 폼의 도구띠

variables - 클라스변수

widget - 폼자체. 이 요소는 다른 창문부품요소들을 비롯한 다른 요소들을 포함할수 있다.

forward - Qt 3.x 베타이전과 호환

include - Qt 2.x 이전과 호환

variable - Qt 3.x 베타이전과 호환

요소들의 순서화는 자유이고 일반적으로 class요소가 우선이다.

1. UI요소

1) actions

이 요소는 폼의 작용들을 보관하는데 쓰인다. 이것은 QMainWindow폼들에서만 발생한다.

actions요소는 하나이상의 action요소들을 포함한다. 매개 action요소는 하나이상의 속성들을 포함한다. 매개 속성은 name속성과 datatype요소안에 포함되는 단일값을 포함한다.

```
<actions>
  <action>
    <property  name="name">
      <cstring>fileNewAction</cstring>
    </property>
    <property  name="iconSet">
      <iconset>filenew</iconset>
    </property>
    <property  name="text">
      <string>New</string>
    </property>
    <property  name="menuText">
      <string>&amp;New</string>
```

```
        </property>
        <property  name="accel">
            <number>4194382</number>
        </property>
    </action>
    ...
</actions>
```

2) author

이 요소는 저자이름을 단순문자렬로 보관하는데 쓰인다.

```
<author>Barney Rubble</author>
```

3) class

이 요소는 폼의 클라스이름을 단순문자렬로 보관하는데 쓰인다.

```
<class>InsuranceForm</class>
```

4) comment

이 요소는 주석 실례로 저작권을 단순문자렬로 보관하는데 쓰인다.

```
<comment>
*****************************************************************
** Copyright (C) 2002 Trolltech AS.  All rights reserved.
** This file is part of Qt Designer.
** This file may be distributed and/or modified under the terms of the
** GNU GPL version 2 as published by the Free Software
** Foundation and appearing in the file LICENSE.GPL included in the
** packaging of this file.
** This file is provided as is with no warranty of any kind,including the
** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A
** PARTICULAR PURPOSE.
** See http://www.trolltech.com/gpl/ for GPL licensing information.
** Contact info@trolltech.com if any conditions of this licensing are
** not clear to you.
*******************************************************************
</comment>
```

5) connections

이 요소는 폼에서 신호-처리부련결을 기록하는데 쓰인다.

connections요소는 하나이상의 connection요소와 하나이상의 slot요소를 포함한다. 매

개 connection요소는 신호하는 객체와 그 신호 그리고 수신하는 객체와 그 처리부를 식별한다.

```
<connections>
  ...
  <connection  language="C++">
    <sender>alignActionGroup</sender>
    <signal>selected(QAction*)</signal>
    <receiver>EditorForm</receiver>
    <slot>changeAlignment(QAction*)</slot>
  </connection>
  ...
</connections>
```

6) customwidgets

Qt Designer는 사용자정의창문부품을 만들수 있다. 사용자정의창문부품은 .ui파일에서 <customwidget>요소에 의하여 표시된다.

이것들은 폼우에 재색직4각형으로 표시되는 낡은 형식의 사용자정의창문부품들이다. 플라그인들을 사용하여 자기의 사용자정의창문부품들을 Qt Designer에 원만히 통합할수 있다(6절 2. 참고).

매개 사용자정의창문부품은 클라스이름과 머리부파일을 가진다. 또한 크기암시와 크기정책을 가진다. 픽스매프가 지정될수 있으며 이것은 사용자가 사용자정의창문부품의 실례를 창조하는데 사용할수 있는 Qt Designer도구띠단추에 표시된다. 사용자정의창문부품들은 보통 신호를 발생하는데 이것들이 렬거된다. 창문부품이 가지는 속성의 이름과 형이 역시 포함된다.

```
<customwidgets>
  <customwidget>
    <class>StyledButton</class>
    <header  location="local">styledbutton.h</header>
    <sizehint>
      <width>40</width>
      <height>25</height>
    </sizehint>
    <container>0</container>
    <sizepolicy>
      <hordata>5</hordata>
```

```
        <verdata>5</verdata>
      </sizepolicy>
      <pixmap>image0</pixmap>
      <signal>clicked()</signal>
      <signal>changed()</signal>
      <property  type="Color">color</property>
      <property  type="Pixmap">pixmap</property>
      <property  type="Bool">scale</property>
    </customwidget>
  </customwidgets>
```

7) exportmacro

이 꼬리표는 Windows사용자와만 관련되여있다.

Windows에 고유한 반출마크로를 요구하는 클라스(실례로 DLL에서 class win_specific_declaration_goes_here와 같이 선언해야 할 클라스)가 있다면 <exportmacro>꼬리표를 리용할수 있다.(표준Qt에서는 Q_EXPORT마크로를 사용한다. 실례로 class Q_EXPORT QWidget.) 이 꼬리표를 사용한다면 또한 다음과 같이 해야 한다.

마크로정의를 포함하는 파일을 포함해야 한다.

폼에 반출마크로를 추가한다 . 이것은 폼의 이름속성의 export macro보조등록부에서 마크로의 이름을 입력하여 달성된다.

다음것은 이 걸음들에 의하여 uic가 정확한 class YOUR_MACRO Form선언들을 창조한다는것을 담보한다.

```
<exportmacro>EDITOR_EXPORT</exportmacro>
```

8) forwards

앞방향선언이 필요한 경우가 있다. 특히 Qt Designer에 의하여 .ui.h 파일들에 코드를 쓰고있다면 더하다. 매개의 앞방향선언은 생성된 C++코드에서 나타나야 하는것처럼 렬거된다.

```
<forwards>
  <forward>class  QStringList;</forward>
</forwards>
```

9) functions

functions요소는 함수목록을 포함한다. 이것은 표준C++함수이다. 함수기호를 창조하기 위하여 각종 속성들을 지정할수 있다. 실례로

access: private, protected 혹은 public를 들수 있다. 기정은 public이다.

returnType: 함수의 돌림값 자료형. 기정은 void

specifier: non virtual, virtual 혹은 pure virtual이다. 기정은 virtual이다.

language: 프로그람언어를 지정한다. 기정은 C++이다.

```
<functions>
  <function access="private" specifier="non virtual">isValid()</function>
  <function access="public" returnType="QString">getMessage()</function>
</functions>
```

10) images

화상은 보통 자체의 파일들에 보관되고 프로젝트파일에 의하여 폼과 련관된다. 이것은 화상들을 하나의 프로젝트에서 그리고 프로젝트들사이에서 임의의 개수의 폼들을 서로 엇바꾸어 공유할수 있다는 우점을 가진다.

일부 경우에 폼안에 직접 화상자료를 보관할것을 요구할수 있으며 <image>끄리표가 이것을 허용한다.

PNG와 XPM형식으로 화상들을 보관할수 있다. 실천에서는 일부 PNG서고들의 오유로 인하여 XPM만 권고된다. 화상자료를 부호화하는 방법을 알려면 resource.cpp안의 saveImageData함수를 참고하시오.

```
<images>
  <image name="image0">
    <data format="XPM.GZ" length="409">789cd3d7528808f055d0d2e72a2e492cc94c5648c
e482c52d04a29cdcdad8c8eb5ade652323453002263054325 1d2e253d856405bffcbc54105b19c85636
0003103711c6b53006ab440370316528264b4c198450c5808a94d1ed00aac214832b43124b544ec414
d34b4c4c441103f113411208313097583 13d0cf3b0840b7258d55a73010092c14eca</data>
  </image>
</images>
```

11) includes

흔히 .ui파일에 머리부파일들을 포함할 필요가 있다. 머리부파일들은 국부적(즉 프로젝트의 등록부에 관하여)이거나 대역적(즉 Qt 혹은 콤파일러표준서고들의 부분)일수 있다. 머리부파일들은 될수록 실현에서 선언된다. 물론 선언(머리부)파일에서 그것들을 선언할 필요가 있다.

Qt Designer는 자동적으로 .ui의 .ui.h 파일에 자동적으로 <include>끄리표들을 추가한다.

```
<includes>
  <include location="local" impldecl="in implementation">pixmapcollection.h</include>
  <include location="local" impldecl="in implementation">pixmapchooser.h</include>
  <include location="local" impldecl="in implementation">project.h</include>
```

```
   <include location="global" impldecl="in implementation">qfileinfo.h</include>
   <include location="global" impldecl="in implementation">qimage.h</include>
   <include location="global" impldecl="in declaration">qpixmap.h</include>
   <include location="local" impldecl="in implementation">pixmapcollectioneditor.ui.h</include>
</includes>
```

12) layoutdefaults

매개의 폼은 기정공간과 여백크기를 가진다. 이것들은 경우에 따라 바꿀수 있다.

```
<layoutdefaults spacing="6" margin="11"/>
```

13) layoutfunctions

흔히 공간과 여백값들은 동적으로 결정되여야 한다. 그래야 례를 들면 창문관리기에서 일반적형식을 줄수 있다.

```
<layoutfunctions spacing="LayoutClass::spacing" margin="LayoutClass::margin"/>
```

margin와 spacing값들에 대한 규칙들은 다음과 같다.

① margin이나 spacing에 대하여 옹근수값이 지정되면 그 값이 사용된다.

② 비옹근수값이 지정되고 배치함수가 지정되면 그 함수가 사용된다.

③ 비옹근수값이나 함수가 지정되면 layoutdefaults로부터 기정값이 사용된다.

14) menubar

QMainWindow를 사용하는 응용프로그람들은 흔히 차림표띠를 가진다. 차림표띠는 이름속성과 하나이상의 튀여나오기차림표항목을 가진다. 매개 차림표항목은 하나이상의 작용들과 분리선을 가진다.

```
<menubar>
   <property name="name">
      <cstring>menubar</cstring>
   </property>
   <item text="&amp;File" name="PopupMenu">
      <action name="fileSaveAction"/>
      <separator/>
      <action name="fileExitAction"/>
   </item>
   <item text="&amp;Help" name="PopupMenu_2">
      <action name="helpAboutAction"/>
      <action name="helpAboutQtAction"/>
   </item>
</menubar>
```

15) pixmapfunction

화상들은 보통 프로젝트파일들에서 자기의 파일이름들을 렬거하여 포함된다. 화상들은 또한 images끄리표에 의하여 직결로 포함될수 있다. 화상들을 취급하는 다른 방법은 함수이름을 지정하는것이다. 이 함수는 련관된 화상의 이름(혹은 열쇠)을 리용하여 호출되며 적당한 화상의 적재에 응답할수 있다. 그러한 함수이름을 지정하려면 <pixmapfunction>끄리표가 사용된다.

```
<pixmapfunction>splashScreen</pixmapfunction>
```

16) pixmapinproject

대부분의 응용프로그람들은 자기의 화상들을 응용프로그람의 프로젝트파일에 목록된 개별적파일들로 보관한다. 이것은 <pixmapinproject>끄리표를 포함하여 표시될수 있다.

```
<pixmapinproject/>
```

17) signals

모든 신호는 <signals>끄리표안에 렬거된다.

```
<signals>
   <signal>somethingChanged()</signal>
</signals>
```

18) slots

slots요소는 처리부목록을 포함한다. 매개의 slot요소는 처리부의 원형을 준다.

```
<slots>
   <slot access="public" specifier="virtual" language="C++"
      returnType="void">changeAlignment(QAction* align)</slot>
</slots>
```

19) tabstops

타브중지는 폼에서 사용자타브로서 초점을 얻는 창문부품을 가리킨다. <tabstops>끄리표는 타브중지목록을 포함하며 매개 타브중지는 창문부품의 이름을 보관한다.

```
<tabstops>
   <tabstop>templateView</tabstop>
   <tabstop>helpButton</tabstop>
   <tabstop>buttonOk</tabstop>
   <tabstop>buttonCancel</tabstop>
</tabstops>
```

20) toolbars

도구띠(류동창문)를 가지는 폼들은 <toolbars>끄리표에 의하여 그 세부를 보관한다. 이 폼들은 보통 QMainWindow들(혹은 파생클라스들)이다. 매개 도구띠는 그 도구띠가

처음에 속하는 류동창문을 식별하는 류동속성을 가진다. 그것들은 또한 이름과 표식자 속성들을 가진다. 매개 도구띠단추는 작용에 의하여 표시된다. 도구띠들은 또한 다른 창문부품들을 보유할수 있다. 이 경우에 <toolbar>끄리표는 련관된 창문부품의 클라스와 이름, 비기정속성값들을 주는 적당한 <widget>끄리표들을 포함한다.

```
<toolbars>
  <toolbar dock="2">
    <property name="name">
      <cstring>toolBar</cstring>
    </property>
    <property name="label">
      <string>Tools</string>
    </property>
    <action name="fileNewAction"/>
    <action name="fileOpenAction"/>
    <action name="fileSaveAction"/>
    <separator/>
    <action name="editUndoAction"/>
    <action name="editRedoAction"/>
    <action name="editCutAction"/>
    <action name="editCopyAction"/>
    <action name="editPasteAction"/>
  </toolbar>
  <toolbar dock="2">
    <property name="name">
      <cstring>Toolbar</cstring>
    </property>
    <property name="label">
      <string>Toolbar</string>
    </property>
    <action name="leftAlignAction"/>
    <action name="centerAlignAction"/>
    <action name="rightAlignAction"/>
    <separator/>
    <action name="boldAction"/>
```

```
        <action name="italicAction"/>
        <action name="underlineAction"/>
        <separator/>
        <widget class="QComboBox">
            <property name="name">
                <cstring>fontComboBox</cstring>
            </property>
        </widget>
        <widget class="QSpinBox">
            <property name="name">
                <cstring>SpinBox2</cstring>
            </property>
            <property name="minValue">
                <number>6</number>
            </property>
            <property name="value">
                <number>10</number>
            </property>
        </widget>
    </toolbar>
</toolbars>
```

21) variables

모듈변수들은 <variables>꼬리표에 보관된다. 변수형이름들은 흔히 <와 >를 포함하며 실체로서 보관되여야 한다. 또한 호출형 즉 public, protected 혹은 private를 지정할수 있다. 기정은 protected이다.

```
<variables>
    <variable access="private">SettingsDialog * settings;</variable>
    <variable>QMap&lt;int, QString&gt; bookmarks;</variable>
    <variable>HelpWindow *browser;</variable>
    <variable>HelpDialog *helpDock;</variable>
    <variable>QGuardedPtr&lt;FindDialog&gt; findDialog;</variable>
    <variable>static QPtrList&lt;MainWindow&gt; *windows;</variable>
</variables>
```

22) widget

창문부품들은 .ui파일의 여러준위에서 사용된다. 전체 폼자체가 창문부품이고 hboxes와 vboxes, grids와 같은 배치문맥내에서 다른 창문부품들을 포함한다.

아래에 완성된 .ui파일의 실례가 있다. 폼자체는 여러가지 비기정속성모임을 가진 QWidget이다. 이 창문부품은 하나의 hbox를 포함하며 이 수평칸은 일부 비기정속성들을 가지며 하나의 QTextBrowser창문부품을 포함한다.

```
<!DOCTYPE UI><UI version="3.1" stdsetdef="1">
<class>WinIntroPage</class>
<widget class="QWidget">
  <property name="name">
    <cstring>WinIntroPage</cstring>
  </property>
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>387</width>
      <height>228</height>
    </rect>
  </property>
  <property name="caption">
    <string>Form1</string>
  </property>
  <hbox>
    <property name="name">
      <cstring>unnamed</cstring>
    </property>
    <property name="margin">
      <number>11</number>
    </property>
    <property name="spacing">
      <number>6</number>
    </property>
    <widget class="QTextBrowser">
```

```
      <property name="name">
        <cstring>TextBrowser1</cstring>
      </property>
      <property name="text">
        <string>This program installs Qt.</string>
      </property>
    </widget>
  </hbox>
</widget>
<layoutdefaults spacing="6" margin="11"/>
</UI>
```

23) forward

이 꼬리표는 Qt 3.x베타이전과 호환되는것들에 포함되며 사용하지 말아야 한다. 대신에 forwards를 사용하시오.

24) include

이 꼬리표는 Qt 2.x베타이전과 호환되는것들에 포함되며 사용하지 말아야 한다. 대신에 includes를 사용하시오.

25) variable

이 꼬리표는 Qt 3.x베타이전과 호환되는것들에 포함되며 사용하지 말아야 한다. 그 대신에 variables를 사용하시오.

2. 자료형요소

bool - 론리값 (0 혹은 1), 실례로 <bool>1</bool>

color - 색, 실례로 <color><red>192</red><green>0</green><blue>255</blue></color>

cstring - C문자렬값 (8-bit), 실례로 <cstring>Some text</cstring>

cursor - 유표형을 가리키는 옹근수, 실례로 <cursor>4</cursor>. 유표형의 유효옹근수는 다음과 같다.

0 - ArrowCursor

1 - UpArrowCursor

2 - CrossCursor

3 - WaitCursor

4 - IbeamCursor

5 - SizeVerCursor

6 - SizeHorCursor

7 - SizeBDiagCursor

8 - SizeFDiagCursor

9 - SizeAllCursor

10 - BlankCursor

11 - SplitVCursor

12 - SplitHCursor

13 - PointingHandCursor

14 - ForbiddenCursor

enum - enum이름, 실례로 <enum>StrongFocus</enum>

font - 서체서술, 실례로

```
<font>
  <family>Helvetica</family>
  <pointsize>16</pointsize>
  <weight>50</weight>
  <italic>1</italic>
  <underline>0</underline>
  <strikeout>0</strikeout>
</font>
```

iconset - 그림기호모임(픽스매프참고), 실례로 <iconset>filenew</iconset>

number - 부호를 가질수 있는 옹근수, 실례로 <number>947</number>

palette - 조색판

pixmap - 픽스매프, 보통 픽스매프의 이름 혹은 열쇠; 이름은 픽스매프들이 프로젝트에 보관되면 사용되고 열쇠는 사용자정의함수가 픽스매프호출에 사용되는 경우에 리용된다. 또한 픽스매프들을 즉시 포함하는것이 가능하다. 실례로 <pixmap>chair</pixmap>

point - 점, 실례로 <point><x>15</x><y>95</y></point>

rect - 직4각형, 실례로

```
<rect>
  <x>20</x>
  <y>35</y>
  <width>225</width>
  <height>45</height>
</rect>
```

set - |에 의해 구분된 이름목록, 실례로 <set>AlignLeft|AlignTop</set>

size - 크기, 실례로 <size><width>150</width><height>105</height></size>

sizepolicy － 크기형을 가리키는 옹근수, 실례로 <hsizetype>5</hsizetype><vsizetype>4</vsizetype>. 크기형의 유효옹근수들은 다음과 같다.

0 - Fixed

1 - Minimum

3 - MinimumExpanding

4 - Maximum

5 - Preferred

7 - Expanding

string - Unicode문자렬값 (UTF8), 실례로 <string>Some text</string>

복합자료형요소들

palette

이 원소는 매개 색그룹에 대하여 사용자대면부요소들을 위한 색을 보유한다. 실례로

```
<palette>
  <active>
  <color> ... Foreground   ... </color>
  <color> ... Button       ... </color>
  <color> ... Light        ... </color>
  <color> ... Midlight     ... </color>
  <color> ... Dark         ... </color>
  <color> ... Mid          ... </color>
  <color> ... Text         ... </color>
  <color> ... BrightText   ... </color>
  <color> ... ButtonText   ... </color>
  <color> ... Base         ... </color>
  <color> ... Background   ... </color>
  <color> ... Shadow       ... </color>
  <color> ... Highlight    ... </color>
  <color> ... HighlightText ... </color>
  </active>
  <disabled>
  <color><red>128</red><green>128</green><blue>128</blue></color>
  ...
  <color><red>255</red><green>255</green><blue>255</blue></color>
  </disabled>
```

<inactive>
<color><red>0</red><green>0</green><blue>0</blue></color>
...
<color><red>255</red><green>255</green><blue>255</blue></color>
</inactive>
</palette>

제2장. Qt번역도구

Qt는 여러 언어로 응용프로그람을 번역하기 위한 우수한 기능을 제공한다. 이 장에서는 응용프로그람의 번역과 관련된 Qt번역도구의 사용방법을 설명한다.

제1절은 응용프로그람개발책임자를 위한것이다. 일반적으로 이것은 쏘프트웨어기사와 번역프로그람의 작업과 일치한다. 여기서는 2개 도구의 사용법을 서술한다. lupdate도구는 원천코드와 번역을 동기시키기 위해 사용된다. lrelease도구는 출하되는 응용프로그람에서 사용하는 실행시번역파일을 만드는데 사용된다.

제2절은 번역프로그람용으로서 Qt Linguist도구의 사용법을 서술한다. 프로그람을 시작하고 본문편집기 또는 문서처리기를 사용하는데는 어떤 콤퓨터지식도 필요없다.

제3절은 Qt프로그람작성자를 위한것으로서 번역본문을 사용할수 있는 Qt응용프로그람을 만드는 방법을 설명한다. 그것도 번역프로그람이 구가 나타나는 문맥을 확인하도록 방조하는 방법의 차림표를 제공한다.

1. 번역과정의 개관

응용프로그람에서 번역되어야 하는 대부분의 본문은 단어나 짧은 구로 이루어져있다. 이것들은 일반적으로 창문제목, 차림표항목, 튀여나오기방조본문(balloon help)과 단추, 검사칸, 라지오단추에 대한 표식자로서 나타난다.

구는 단순하면서 특별한 문법을 사용하는 자국어로 프로그람작성자에 의해 원천코드에 입력된다. 문법은 번역을 요구하는 구를 식별한다. Qt도구는 번역프로그람를 도와주기 위하여 매개 구에 대한 문맥정보를 제공한다. 그리고 프로그람작성자는 필요하다면 구에 문맥정보를 추가할수 있다. 출하판관리기는 원천파일로부터 만들어지는 한조의 번역파일을 생성하여 번역프로그람에게 넘긴다. 번역프로그람은 Qt Linguist를 사용하여 번역파일을 열고 번역을 입력하고 번역파일들에 결과를 내보내며 그것을 출하판관리기에 넘긴다. 출하판관리기는 그다음 응용프로그람에서 사용할 준비가 되여있는 번역파일들의 고속이고 조밀한 판을 생성한다. 응용프로그람을 변경하고 판갱신하는 반복되는 주기에서 현존번역을 보존하고 새로운 번역이 요구된다는것을 쉽게 식별할수 있도록 도구를 설계한다. Qt Linguist도 여러개의 응용프로그람과 프로젝트들에 관하여 일관적인 번역을 정확히 하도록 돕기 위해 숙어집의 편리를 제공한다.

인간언어의 미묘성과 복잡성때문에 번역프로그람과 프로그람작성자는 많은 문제에 대답해야 한다.

- 하나의 구는 문맥에 따라 여러가지 형태로 번역될수 있다. 례를 들면 영어의 open은 도이췰란드어로 öffnen, "open file" 또는 aufbauen, "open internet connection"으로 될수 있다.

• 지름건이 충돌하지 않게 변경할수 있어야 한다. 례를 들면 영어로 "&Quit"는 "Q"를 포함하지 않는 노르웨이어의 "Avslutt"로 된다. 우리는 이미 사용하고있는 문자를 사용할수 없으며 몇개의 지름건을 변경해야 한다.

• 변수를 포함하는 구, 례를 들면 《선택된 25개의 파일을 처리하는데 63s 걸린다》에서는 실행시에 2개의 수를 프로그람적으로 삽입하며 다른 언어에서는 어순과 변수의 배치가 달라야 할수 있으므로 바꾸어 서술해야 할수 있다.

Qt번역도구는 이 문제를 간단하게 해결한다.

제1절. 출하판관리기

출하판관리기로서 2개의 도구 lupdate와 lrelease가 제공된다. 이 도구는 qmake 프로젝트파일에 의존한다. 그러나 반드시 qmake를 사용하는것은 아니다.

세번째도구 qm2ts는 Qt 2.x통보문파일을 .ts파일로 변환하는데 사용할수 있다.

1. Qt프로젝트파일

lupdate와 lrelease는 응용프로그람의 .pro Qt프로젝트파일의 정보에 의존한다. 자국어에 추가되는 매개 언어에 대하여 프로젝트파일의 TRANSLATIONS절에 항목이 있어야 한다. 전형적인 항목은 다음과 같다.

```
TRANSLATIONS    = tt2_fr.ts \
                  tt2_nl.ts
```

번역파일이름안에서 지역을 사용하면 실행시에 적재해야 할 언어를 결정하는데 도움이 된다. 이것은 3절에서 설명한다.

4개의 번역원천파일이 있는 완전한 .pro파일의 례:

```
HEADERS        =  main-dlg.h  \
                  options-dlg.h
SOURCES        =  main-dlg.cpp  \
                  options-dlg.cpp  \
                  main.cpp
FORMS          =  search-dlg.ui
TRANSLATIONS   =  superapp_dk.ts  \
                  superapp_fi.ts  \
                  superapp_no.ts  \
                  superapp_se.ts
```

QTextCodec::setCodecForTr()는 tr()호출내에서 나타나는 상수문자렬을 위한

8bit부호화를 선택할수 있게 한다. 이것은 원천언어가 례를 들면 중국어이거나 일본어인 응용프로그람에 도움이 된다. 만일 어떤 부호화도 설정하지 않으면 tr()는 Latin-1을 사용한다.

만일 응용프로그람에서 QTextCodec::codecForTr()기구를 사용하면 Qt Linguist는 .pro파일안의 DEFAULTCODEC항목도 설정할것을 요구한다. 실례로

DEFAULTCODEC = ISO-8859-5

2. lupdate

사용법: lupdate myproject.pro

이것은 간단한 지령행도구이다. lupdate는 Qt .pro프로젝트파일을 읽어들이고 지정된 원천, 머리부와 Qt Designer대면부파일안의 번역가능한 문자렬들을 검색하고 프로젝트파일에 목록된 .ts번역파일을 생성하고 갱신한다. 번역파일은 Qt Linguist가 파일을 읽어들여 번역에 삽입하는데 사용하는 번역프로그람에 주어진다.

자체의 번역프로그람을 가지고있으면 응용프로그람을 개발할 때 정기적으로(매달) lupdate를 실행하는것이 유용하다는것을 알고있다. 이것은 프로젝트에 대한 번역작업이 아주 적어지게 하고 번역프로그람이 많은 프로젝트를 동시에 지원하게 한다.

필요할 때에 번역프로그람을 임대받는 회사들은 응용프로그람의 생명주기에서 극히 얼마 안되는 시간만 lupdate를 실행하는것이 좋을수 있다. 이것은 번역프로그람에 본질적인 단일블로크의 작업을 제공하며 번역프로그람이 찾는 어떤 오유라도 처음의 시험단계에서 발견하기 쉽게 한다. 두번째와 그후의 lupdate는 아마 최종적인 베타단계에서 수행된다.

.ts파일형식은 필요하다면 판조종체계와 함께 사용할수 있고 사람이 읽을수 있는 단순한 XML형식이다.

3. lrelease

사용법: lrelease myproject.pro

이것은 또 하나의 간단한 지령행도구이다. 그것은 Qt .pro프로젝트파일을 읽어들이고 프로젝트파일에 목록되는 각 .ts번역원천파일에 대하여 응용프로그람에서 사용하는 .qm파일을 생성한다. .qm파일형식은 상당히 빠른 검색을 번역에 제공하는 압축된 2진형식이다.

이 도구는 초기의 시험판으로부터 최종적인 출하판까지 응용프로그람을 만들 때 항상 실행된다. .qm파일들이 만들어지지 않으면 례를 들어 번역에 착수하기전에 알파판의 발표가 요구되기때문에 프로그람작성자가 원천파일에 배치된 본문을 사용하여 응용프로그람이 아주 잘 실행되게 한다. .qm파일들이 유효하면 응용프로그람이 그것들을 찾아서 자동적으로 사용한다.

lrelease는 반드시 Done으로 표식된 번역만 포함된다. 만일 번역이 실패하거나 확증에서 실패하면 원시본문이 그대신에 사용된다.

4. 번역실패

lupdate와 lrelease는 둘다 완전하지 않은 .ts번역원천파일과 사용될수 있다. 번역실패는 실행시에 자국어의 문구로 교체된다.

제2절. 번역기

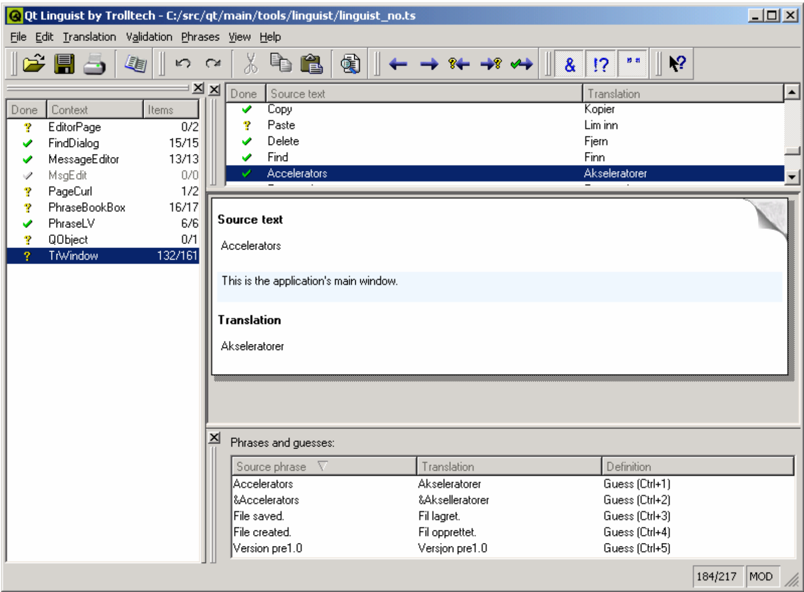


그림 2-1. Linguist의 기본창문

1. Qt Linguist사용에 관한 간단한 차림표

Qt Linguist는 Qt응용프로그람에 번역을 추가하기 위한 도구이다. 여기서는 화면에(실례로 같은 차림표나 대화칸에) 함께 나타나는 한 그룹의 구들을 의미하는 번역문맥의 개념을 소개한다.

과제띠차림표로부터 혹은 탁상그림기호를 두번 련속 누르고 지령행에서 linguist(이어서 Enter)라고 입력하여 Qt Linguist를 기동한다. Qt Linguist가 기동하면 차림표띠에서 File|Open를 선택하고 작업하려는 .ts번역원천파일을 선택한다.

Qt Linguist의 기본창문은 4개의 기본구역으로 분할된다. 왼쪽에는 Context목록, 오른쪽우에는 Source본문구역, 오른쪽중심에는 번역구역, 오른쪽바닥에는 구와 추측구역이 있다.

문맥목록(왼쪽)에서 문맥중의 하나를 선택하고 Source본문구역(오른쪽우)에 나타

나는 구중의 하나를 선택한다. 구는 번역구역(오른쪽중간)에 복사된다. 단어 'Translation'아래를 선택하고 번역에 입력한다. Ctrl+Enter (Done & Next)를 눌러서 번역을 완료했다는것을 확인하고 번역을 요구하는 다음 구까지 이동한다.

Ctrl+Enter를 누른 다음 번역에 들어가는 주기는 모든 번역이 수행되거나 쎄션을 끝낼 때까지 반복될수 있다. Linguist는 열린 구서적과 이전의 번역들로부터 《구와 추측》구역을 가능한 번역으로 채우려고 한다. 매개는 지름건을 가지고있다. 례를 들면 Ctrl+1, Ctrl+2 등은 번역구역으로 추측을 복사하는데 사용할수 있다. (마우스사용자는 구 혹은 추측을 두번 눌러서 번역구역으로 그것을 움직일수 있다.) 세션의 끝에서 차림표띠로부터 File|Save를 선택한 다음 File|Exit를 눌러서 완료한다.

2. Qt Linguist의 기본창문

1) 문맥목록

이것은 기정으로 기본창문의 왼쪽에 나타난다. 첫째 렬 Done은 문맥을 위한 번역이 수행되였는가를 확인한다. 검사표식은 번역이 끝나고 확증되였다는것을 가리킨다. 물음표는 하나이상의 번역이 수행되지 않았거나 확증에서 실패하였다는것을 가리킨다. 두번째 렬 Context는 번역구가 나타나는 문맥의 이름이다. 셋째 렬 Items은 2개 수를 보여준다. 셋째는 수행된 번역수이고 두번째는 문맥에 있는 구의 개수이다. 수들이 같으면 모든 번역은 끝났다. 회색으로 된 검사표식은 이전의 번역 즉 응용프로그람의 이전판에서 번역되였는데 새로운 판에 번역하지 않은 구를 가리킨다.

문맥은 영자모순으로 정렬된다. 각 문맥안에 있는 구들은 화면우에서 표시되는 순서가 아니라 원천프로그람에 나타나는 순서로 놓인다.

Context List는 류동가능창문이므로 기본창문의 다른 위치에 끌어가거나 기본창문밖으로 끌어가 자기의 오른쪽에 놓을수도 있다. Context List를 움직이면 Qt Linguist는 그 위치를 기억하고 프로그람을 기동할 때마다 그것을 본래상태로 되살린다.

2) 원천본문구역

이것은 기정적으로 기본창문의 오른쪽우에 나타난다. 첫째 렬('Done')은 번역상태를 보여준다. 검사표식은 구가 번역되고 확증되였다는것을 가리킨다. 물음표는 번역이 완성되지않았다는것을 가리킨다. 느낌표는 번역이 확증에서 실패하였다는것을 가리킨다. 둘째 렬 'Source text'는 번역되여야 하는 본문을 보여준다. 셋째 렬은 번역을 보여준다.

Qt Linguist는 3종류의 확증을 제공한다. 즉 지름기호(accelerator), 구두점(punctuation)과 구이다. 만일 원천본문이 지름기호 즉 &이고 번역된 본문이 지름기호(즉 &)를 포함하지 않으면 번역은 지름기호확증에 실패한다. 마찬가지로 원천본문이 특수구두점 례하면 ?, ! 또는 .이고 번역이 각이한 구두점으로 끝나면 번역은 구두점확증에 실패한다. 사용된 번역과는 다른 열린 성구집의 하나에 번역이 있으면 번역은 구두점

확증에 실패한다.

원천본문령역은 류동가능창문이다.

3) 번역구역

번역구역(Translation area)은 기정으로 기본창문의 오른쪽 중심에 나타난다. 여기에는 3개의 수직구간들이 포함된다. 첫째 구간은 Source text라고 표식되여있고 그 아래에 원천본문이 나타난다. 둘째 구간에는 프로그람작성자가 번역기를 방조하기 위하여 추가한 연청색배경의 문맥정보가 포함된다. 문맥정보가 주어지지 않았다면 이 구간은 나타나지 않는다. 셋째 구간은 Translation으로 표식되여있고 여기는 원천본문의 번역을 써넣는 곳이다.

4) 구 및 추측구역

이 구역은 기정으로 기본창문의 오른쪽 아래에 나타난다. 새 구로 이동할 때 그 구가 적재되여있는 성구집들중 하나에 있으면 구가 이 구역안에 그 번역문과 함께 나타난다. 구가 이미 번역된 다른 구와 같거나 비슷하다면 구와 번역문이 이 구역에 나타난다. 구 및 추측구역에서 번역문을 복사하려면 F6을 눌러서 구 및 추측구역으로 옮기고 올리와 내리화살건에 의하여 사용하려는 구로 옮기고 Enter를 눌러서 복사한다. 후에 구를 복사하지 않으려고 결심하였다면 Esc를 누른다. 두 경우에 초점은 번역구역으로 돌아온다. 또한 사용하려는 번역문을 두번 찰칵하여 번역구역에로 복사한다.

구 및 추측구역(Phrases and Guesses Area)은 류동가능창문이다.

3. 공동과제

1) 후번역

번역을 그만두려고 한다면 Ctrl+L (Next Unfinished)을 눌러서 다음의 완료되지 않은 번역으로 이동한다. 완료되지 않은 번역은 전혀 번역되지 않았거나 확증에서 실패한것이다. 다음의 구로 이행하려면 Shift+Ctrl+L을 누른다. 또한 Translation차림표를 리용하여 항행할수 있다. 완전히 다른 문맥으로 이행하려면 Context목록에서 작업하려는 문맥을 찰칵하고 Source Text구역에서 원천본문을 찰칵한다.

2) 문맥에 따르는 여러개의 번역을 요구하는 구

같은 구가 2개이상의 문맥에서 충돌없이 나타날수 있다. 일단 한개 문맥에서 구가 번역되였으면 Qt Linguist는 그것이 이미 번역되였다는것을 말하며 후에 번역기가 꼭같은 구와 만날 때 Qt Linguist는 구와 추측구역에서 이전의 번역문을 가능한 번역문후보로서 제공한다. 이전의 번역문을 받아들일수 있으면 Done & Next 단추를 찰칵하여 (Alt+Enter를 눌러서) 다음의 완료되지 않은 구로 이동한다.

어떤 구가 특정문맥에 한번이상 발생하면 그것은 Qt Linguist의 문맥목록에 오직 한번 표시되며 그 번역은 그 문맥안의 모든 출현에 적용된다. 같은 구가 같은 문맥내에서 다르게 번역되여야 한다면 프로그람작성자는 련관된 모든 구들에 대하여 각이한 주해

를 제공해야 한다. 그러한 주해들이 사용된다면 중복되는 구들이 문맥목록에 나타난다. 프로그람작성자들의 주해는 연청색배경우의 번역구역에 나타난다.

3) 지름건변경

지름건(keyboard accelerator)은 눌렀을 때 응용프로그람이 작용을 수행하게 하는 건결합이다. 보통 지름건은 Alt건과 Ctrl건형식의 지름건이다.

Alt지름건은 차림표와 단추들에 사용된다. 밑줄은 밑줄이 있는 문자를 Alt건과 함께 누르는것이 마우스로 차림표항목을 찰칵하는것과 같다는것을 의미한다. 실례로 대다수 응용프로그람들은 "file"단어의 "F"에 밑줄이 있는 File차림표를 가지고있다. 이러한 응용프로그람들에서 파일차림표는 차림표띠의 "File"단어를 찰칵하거나 Alt+F건을 눌러서 펼칠수 있다. 밑줄있는 지름건은 그 앞에 &기호를 배치하여 지정한다. 실례로 &File을 들수 있다. 원천구가 &와 함께 나타나면 번역문도 같은 문자의 앞에 &를 포함할수 있다. Alt지름건의 의미는 &가 매몰되는 구로부터 결정될수 있다. 번역기는 Alt건과 함께 사용된 문자를 변경할 필요가 있을수 있다. 가령 번역된 구가 원시지름문자를 포함하지 않는 경우이다. 다른 건들과의 충돌(즉 같은 문맥에서 같은 문자를 사용하는 두개의 Alt지름건이 있을 때)은 피하여야 한다. 일부 Alt지름건(보통 차림표띠우에 있는것들)을 다른 문맥에서 적용할수 있다.

Ctrl지름건들은 시각적인 조종요소와 독립적으로 존재할수 있다. 이것들은 흔히 여러개의 건치기나 마우스찰칵을 가지는 차림표의 작용들을 호출하는데 쓰인다. 또한 차림표나 단추에 나타나지 않는 작용들을 수행하는데 쓰일수 있다. 실례로 File차림표를 가지는 대부분의 응용프로그람들은 New라고 부르는 보조차림표를 가져야 한다. 대부분의 응용프로그람들에서 이것은 "New... Ctrl+N"로서 나타난다. 이 차림표선택은 마우스로 File을 찰칵하고 New를 찰칵하여 펼칠수 있다. 혹은 Alt+F를 누르고 밑줄이 있는 N을 누를수 있다. 그러나 단순히 Ctrl+Enter를 눌러서 같은것을 달성할수도 있다. Ctrl건을 사용하는 지름건들은 원천본문에 글자그대로(실례로 Ctrl+Enter) 표시된다. Ctrl지름건들은 구를 가지지 않으므로 번역기는 프로그람작성자에게 의거하여 오른쪽웃구획에 나타나는 "comment"을 추가해야 한다. 이 주석은 Ctrl지름건이 수행하는 작용에 대하여 설명해야 한다. 실제로 Ctrl지름건들은 단순히 그것들을 복사하고 Begin from Source단추를 찰칵하여 번역한다. 그러나 일부 경우에 문자가 목표언어에서 의미를 가지지 않으므로 변경되여야 한다. 어떤 문자나 수자가 선택되든 번역은 항상 Ctrl과 대문자나 수자의 결합이여야 한다. Alt지름건들처럼 번역기가 그 건을 변경하는 경우에 다른 Ctrl지름건과 충돌하지 말아야 한다.

4) 변수들을 포함하는 구에 대한 론의

어떤 구들은 변수를 포함한다. 변수들은 실행시에 채워지는 본문항목들을 위한 대리기호이다. 변수들은 원천본문에서 %기호와 수자들의 결합으로 지정된다. 실례로

After processing file %1, file %2 is next in line을 들수 있다. 이 실레에서 %1은 실행시에 처리하려는 첫 파일의 이름으로 교체되고 %2는 처리해야 할 다음 파일의 이름으로 교체된다. 번역판에서 변수들은 여전히 나타난다. 실례로 도이칠란드번역문은 구들을 역전할수 있다. 실례로 Datei %2 wird bearbeitet, wenn. Datei %1 fertig ist. 두 변수가 여전히 사용되지만 그 순서는 변경되였다. 변수들이 나타나는 순서에는 문제가 없다. %1은 늘 실행시에 같은 본문으로 교체되며 원천본문이나 번역문에서 그것이 나타나는 순서에는 문제가 없다. %2도 마찬가지이다.

5) 번역의 재리용

번역된 본문이 원천본문과 류사하다면 Begin from Source단추를 찰칵하여(Alt+T를 눌러서) 원천본문을 번역구역에 복사한다.

Qt Linguist는 자동적으로 열려진 성구집의 구들과 이미 번역되여있는 류사하거나 같은 구들을 구 및 추측구역에 렬거한다.

4. 성구집의 창조와 사용

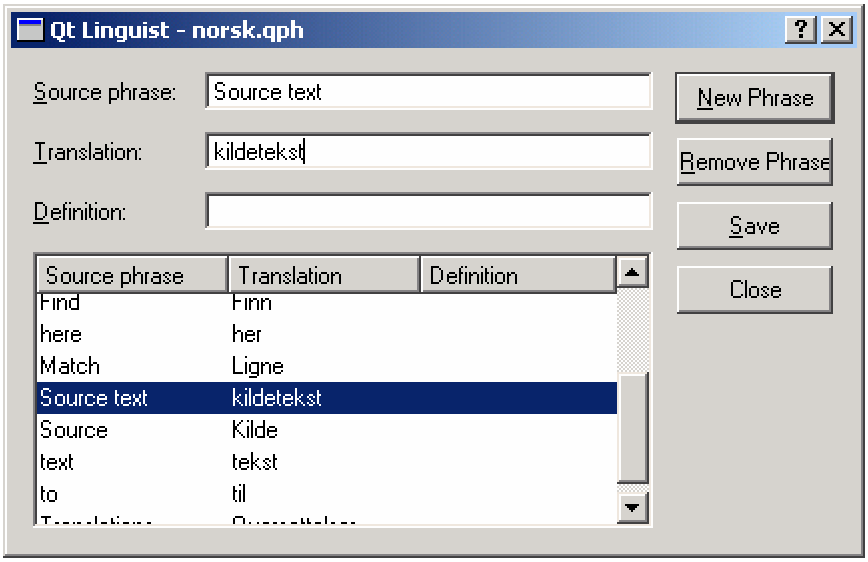


그림 2-2. 성구집대화칸

Qt Linguist성구집은 원천구와 목표(번역된)구, 선택적인 정의들의 모임이다. 비록 응용프로그람이나 한 계렬의 응용프로그람들마다 하나의 성구집이 창조된다하여도 성구집은 응용프로그람과는 독립적으로 창조된다.

번역기가 성구집의 원천구와 같은 번역되지 않은 구에 이르면 Qt Linguist는 기본창문의 오른쪽아래의 Relevant phrases판에 성구집항목을 표시한다. 성구집에서 주어진것들과 충돌하는 번역문들을 가지는 구에는 원천본문구획에서 물음표로 표식된다. 성구집은 번역문들의 공통모임을 제공하여 일관성을 담보하게 하는데 쓰인다. 또한 한 계렬의 응용프로그람들을 위한 번역문들이 성구집에서 일단 생성된 다음 노력의 중복을 피

하는데 쓰이며 성구집은 매개 응용프로그람에서 대부분의 번역문들에 쓰인다.

성구집을 편집하기전에 그것이 창조되여야 하며 또한 그것이 이미 존재한다면 열려져있어야 한다. 차림표띠로부터 Phrase|New Phrase BoOK를 찰칵하여 새로운 성구집을 창조한다. 파일이름을 입력해야 하면 필요하다면 파일의 위치를 바꿀수 있다. 새로 창조한 성구집은 자동적으로 열려진다. 차림표띠로부터 Phrase|Open Phrase BoOK를 찰칵하여 현존성구집을 연다.

새로운 구를 추가하려면 New Phrase단추를 찰칵하고(혹은 Alt+N을 누르고) 새 원천구에 입력한다. Tab를 누르고 번역에 입력한다. Tab를 누르고 정의를 입력한다. 이것은 같은 원천구들의 각이한 번역을 구별하는데 쓸모있다. 이 과정을 필요할 때마다 반복할수 있다.

구목록에서 구를 선택하고 Remove Phrase를 찰칵하여 삭제할수 있다.

Save단추를 찰칵하고(Alt+S를 누르고) 구들을 일단 완료한(삭제한) 다음 Close단추(Esc)를 누른다.

구 혹은 구모임이 성구집에 나타날 때 필요한 목표구를 두번 찰칵하여 본문유표위치의 번역구획에 복사한다. 번역구획안의 본문을 목표구로 교체하려면 번역구획을 찰칵하고 Edit|Select All(Alt+A)을 누른 다음 목표구를 두번 찰칵한다.

5. 확증

Qt Linguist는 번역구들에 대한 3종류의 확증(validation)을 제공한다.

① 지름건확증은 원천구가 수행하는 경우와 반대로 하는 경우에 &를 가지지 않는 번역된 구들을 탐지한다.

② 구두점확증은 원천구와 번역된 구들사이의 마감구두점에서의 차이를 탐지한다. 실례로 원천구가 생략기호, 느낌표 혹은 물음표로 끝나면 경고하지만 번역된 구인 경우에는 경고하지 않는다.

③ 구확증은 성구집에도 있는 원천구들을 탐지하지만 그 번역은 성구집에 주어지는 것과 다르다.

확증은 차림표띠의 Validation항목으로부터 혹은 도구띠단추들을 리용하여 절환한다. 확증에서 실패하는 구들은 원천본문구획에서 물음표로 표식된다. 확증을 차단하고 후에 그것을 설정한다. Qt Linguist는 모든 구들을 다시 검사하고 확증에서 실패한것을 표식한다.

문맥안의 구가 무효이면 문맥자체는 물음표로 표식되며 문맥안의 모든 구가 수행되고 유효이면 그 문맥은 점으로 표식된다.

수행된것으로(점으로) 표식되는 구들만 응용프로그람에서 나타난다. 무효한 구들과 번역되지만 수행한것으로 표시되지 않은 구들은 번역원천파일에 보관되지만 응용프로그람에 의해 사용되지 않는다.

6. Qt Linguist참고

1) 파일형

Qt Linguist는 3종류의 파일을 사용하게 한다. 즉

• .ts번역원천파일은 원천구들과 그 번역문들을 포함하는 사람이 읽을수 있는 XML 파일들이다. 이 파일들은 보통 lupdate에 의해 창조되여 갱신되며 응용프로그람에 고유하다.

• .qm Qt통보문파일은 실행시에 응용프로그람에 의해 사용된 번역문들을 포함하는 2진파일이다. 이 파일들은 lrelease에 의해 생성되지만 Qt Linguist에 의해서도 생성될수도 있다.

• .qph Qt성구집파일은 표준구들과 그 번역문들을 포함하는 사람이 읽을수 있는 XML파일이다. 이 파일들은 Qt Linguist에 의해 창조되고 갱신되며 임의의 수의 프로젝트와 응용프로그람들에서 사용될수 있다.

2) 차림표띠



그림 2-3. 차림표띠

- File

Open... Ctrl+O - 번역원천.ts파일을 선택할수 있는 파일열기대화칸을 펼친다.

Save Ctrl+S - 현재의 번역원천 .ts파일을 보관한다.

Save As... - 파일보관대화칸을 펼치여 현재의 번역원천 .ts 파일을 다른 이름으로 보관하거나 다른 위치에 넣을수 있게 한다.

Release... - 파일보관대화칸을 펼친다. 입력한 파일이름은 현재번역원천파일에 기초한 번역의 Qt통보문.qm 파일이다. 출하판관리기의 지령행도구 lrelease는 응용프로그람의 번역원천파일 모두에 대하여 같은 기능을 수행한다.

Print... Ctrl+P - 인쇄대화칸을 펼친다. OK를 찰칵하면 번역원천과 번역문들이 인쇄된다.

Recently opened files - 최근에 열었던 .ts 파일들을 표시하고 하나를 찰칵하여 열수 있게 한다.

Exit Ctrl+Q - Qt Linguist를 닫는다.

- Edit

Undo Ctrl+Z - 번역구획에서 마지막 편집작용을 취소한다.

Redo Ctrl+Y - 번역구획에서 마지막 편집작용을 재시행한다.

Cut Ctrl+X - 번역구획에서 선택된 본문을 삭제하고 사본을 오려둠판에 보관한다.

Copy Ctrl+C - 번역구획에서 선택된 본문을 오려둠판에 복사한다.

Paste Ctrl+V - 오려둠판본문을 번역구획에 붙이기한다.

Select All Ctrl+A - 번역구획에서 복사 혹은 삭제할 모든 본문을 선택한다.

Find... Ctrl+F - Find대화칸을 펼친다. 대화칸이 펼쳐질 때 검색하려는 본문을 입력하고 Find Next단추를 찰칵한다. 원천구와 번역문, 주석을 검색할수 있다.

Find Next F3 - Find대화칸에서 입력한 본문의 다음번 출현을 찾는다.

- Translation

Prev Unfinished Ctrl+K - 제일 최근의 완료되지 않은 원천구로 이동한다. (완료하지 않았다는것은 번역되지 않았다는것이라든가 번역은 되였으나 확증에서 실패하였다는것을 의미한다.)

Next Unfinished Ctrl+L - 다음의 완료하지 않은 원천구로 이동한다.

Prev Shift+Ctrl+K - 이전의 원천구로 이동한다.

Next Shift+Ctrl+L - 다음의 원천구로 이동한다.

Done & NextCtrl+Enter - 이 구를 번역이 수행된것으로 표식하고 다음의 완료되지 않은 원천구로 이동한다.

Begin from Source Ctrl+B - 원천본문을 번역문에 복사한다.

- Validation

Accelerators - Alt지름건에 대한 확증을 금지 혹은 설정한다.

Ending Punctuation - 구마감구두점 실례로 생략기호, 느낌표, 물음표 등에 대하여 확증을 금지 혹은 설정한다.

Phrase Matches - 현재 성구집안에 있는 번역문들의 대조에 대한 확증을 금지 혹은 설정한다.

- Phrase

New Phrase Book... Ctrl+N - 파일보관대화칸을 펼친다. 성구집에서 사용하려는 파일이름을 입력하고 그 파일을 보관해야 한다. 일단 보관한 다음 성구집을 열고 그것을 사용할수 있다.

Open Phrase Book... Ctrl+H - 파일열기대화칸을 펼친다. 열려는 성구집을 찾고 선택한다.

Close Phrase Book - 현재 성구집을 닫는다. 이것은 성구확증이 더는 발생하지 않게 한다. Validation차림표나 구도구띠단추에 의하여 구확증을 금지함으로써 같은 효과를 얻을수 있다.

Edit Phrase Book... - 성구들을 추가, 편집 혹은 삭제하려는 성구집대화칸을 펼친다.

Print Phrase Book... - 인쇄대화칸을 펼친다. OK를 찰칵하면 그 성구집이 인쇄된다.

- View

Revert Sorting - 원천본문구획안의 구들을 자기의 원래순서로 넣는다.

Display Guesses - 구 및 추측의 현시를 허가 혹은 금지한다.

Statistics - Statistics대화칸의 표시를 절환한다.
Views - Context과 Source본문, Phrase보기들의 표시를 절환한다.
Toolbars - 각이한 도구띠들의 표시를 절환한다.
3) 도구띠



그림 2-4. 도구띠

 새로운 원천 .ts파일을 열기 위하여 파일열기대화칸을 펼친다.

 현재 번역원천 .ts파일을 보관한다.

 현재의 번역원천 .ts파일을 인쇄한다.

 새로운 성구집 .qph파일을 열기 위하여 파일열기대화칸을 펼친다.

 번역구획에서 마지막 편집작용을 취소한다.

 번역구획에서 마지막 편집작용을 재시행한다.

 번역구획에서 선택된 본문을 삭제하고 사본을 오려둠판에 보관한다.

 번역구획에서 선택된 본문을 오려둠판에 복사한다.

 오려둠판본문을 번역구획에 붙이기한다.

 Find대화칸을 펼친다.

 이전의 원천구로 이동한다.

 다음의 원천구로 이동한다.

 이전의 완료되지 않은 원천구로 이동한다.

 다음의 완료되지 않은 원천구로 이동한다.

 이 구를 번역이 수행된것으로 표식하고 다음의 완료되지 않은 원천구로 이동한다.

 Alt지름건에 대한 확증을 금지 혹은 설정한다.

 구마감구두점에 대한 확증을 금지 혹은 설정한다.

 성구집확증을 금지 혹은 설정한다.

4) Find대화칸

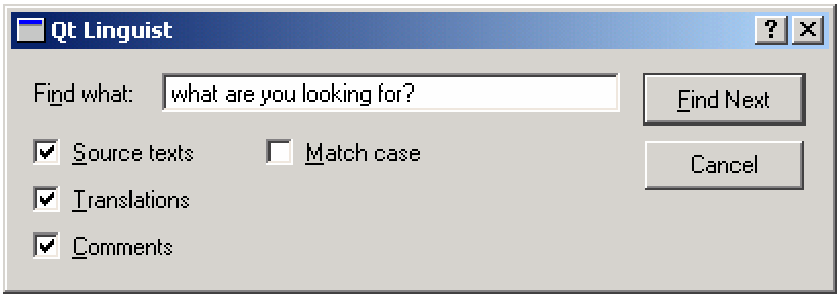


그림 2-5. Find대화칸

차림표띠로부터 Edit|Find를 선택하거나 Ctrl+F를 눌러서 Find대화칸을 펼친다. F3을 눌러서 마지막 검색을 반복한다. 기정으로 원천구와 번역문, 주석들은 모두 검색되며 검색은 대소문자를 구별한다. 이러한 설정은 검사칸들을 검사하거나 검사를 해제하여 자기의 기호에 맞게 변경할수 있다.

5) Phrase대화칸

이 대화칸은 소절 4.에서 설명한다.

제3절. 프로그람작성자

Qt응용프로그람에서 다국어의 유지는 아주 간단하며 프로그람작성자에게 약간한 부담을 요구한다.

Qt는 구들이 만들어지는 각 창문에 대하여 구를 번역하므로 번역사용비용을 최소화한다. 대부분의 응용프로그람들에서 기본창문은 한번만 만들어진다. 대화칸은 보통 창조된 다음 표시되고 필요에 따라서 숨겨진다. 초기의 번역이 진행되였다면 번역된 창문에

대한 부담은 더는 없다. 생성파괴되였다가 다시 생성되는 창문들만 번역비용을 가진다.

Qt에서는 실행시에 언어를 절환할수 있는 응용프로그람을 만들수 있으나 프로그람 작성자의 일정한 간섭을 요구하고 약간의 실행시비용을 요구한다.

1. 응용프로그람번역의 인식

응용프로그람이 적당한 번역파일을 찾아서 적재한 다음 프로그람작성자들은 사용자가 볼수 있는 본문과 Ctrl지름건을 번역목표로 표식한다.

번역을 요구하는 매개 본문은 번역기가 프로그람에서 본문이 있는 위치를 확인할것을 요구한다. 각이한 번역을 요구하는 여러개의 같은 본문인 경우에도 번역기는 원천본문의 번역에 필요한 정보를 요구한다. 번역을 위한 본문선택은 자동적으로 클라스이름을 기초문맥정보로 사용되게 한다. 일부 경우에 프로그람작성자는 번역기를 도와주기 위한 추가정보를 요구할수 있다.

1) 번역파일생성

번역파일들은 응용프로그람에서 사용자가 볼수 있는 모든 본문과 Ctrl건지름기호 그리고 그 본문의 번역으로 이루어진다. 번역파일은 다음과 같이 생성된다.

① 처음에 lupdate를 실행하여 사용자가 볼수 있는 모든 본문만 있고 번역은 없는 .ts번역원천파일의 한조를 생성한다.

② .ts파일들은 Qt Linguist를 사용하여 번역을 추가하는 번역기에게 주어진다. Qt Linguist는 변경되거나 삭제된 어떤 원천본문이라도 처리한다.

③ lupdate를 실행하여 응용프로그람에 추가되는 새로운 본문을 삽입한다. lupdate는 응용프로그람에서 사용자가 볼수 있는 본문을 번역과 동조시키며 어떤 자료도 파괴하지 않는다.

④ 필요하다면 걸음②와 ③을 반복한다.

⑤ 응용프로그람의 발표가 요구될 때 lrelease가 실행되여 .ts파일을 읽고 실행시에 응용프로그람에 의해 사용되는 .qm파일을 생성한다.

lupdate가 성과적으로 작업하기 위하여서는 생성해야 할 번역파일을 알아야 한다. 파일들은 단지 응용프로그람의 .pro Qt프로젝트파일에 목록화된다. 례를 들면

```
TRANSLATIONS   = tt2_fr.ts \
                 tt2_nl.ts
```

2) 번역의 적재

```
int main( int argc, char **argv )
{
    QApplication app( argc, argv );
```

이것은 Qt응용프로그람의 단순한 main()함수를 시작하는 방법이다.

```
int main( int argc, char **argv )
```

```
{
   QApplication app( argc, argv );

   QTranslator translator( 0 );
   translator.load( "tt1_la", "." );
   app.installTranslator( &translator );
```

번역인식응용프로그람을 위하여 번역기객체가 생성되고 번역이 적재되고 응용프로그람에 번역기객체가 설치된다.

```
int main( int argc, char **argv )
{
   QApplication app( argc, argv );

   QTranslator translator( 0 );
   translator.load( QString("tt2_") + QTextCodec::locale(), "." );
   app.installTranslator( &translator );
```

생성응용프로그람에서 더 유연한 수법(례를 들면 지역에 따르는 번역적재)이 적합할수도 있다. 만일 .ts파일이 appname_locale(례를 들면 tt2_fr, tt2_de와 같은 관례를 따라 모두 이름을 지으면 우의 코드는 실행시에 현재지역의 번역을 적재한다.

현재지역용의 어떤 번역파일도 없으면 응용프로그람은 원시원천본문을 사용한다.

3) 응용프로그람이 사용자가 볼수 있는 문자렬을 번역하기

사용자가 볼수 있는 문자렬은 tr()호출안에 그것을 넣어서 번역목표로 표식한다. 례를 들면

```
button = new QPushButton( "&Quit", this );
```

은 다음과 같아진다.

```
button = new QPushButton( tr("&Quit"), this);
```

Q_OBJECT마크로를 사용하는 모든 QObject보조클라스는 tr()함수를 실현한다.

보통 tr()가 QObject보조클라스의 성원함수로서 호출되므로 그 호출이 직접 이루어진다 하더라도 다른 경우에 명시적인 클라스이름을 제공할수 있다. 례를 들면

```
QPushButton::tr("&Quit")
```

혹은

```
QObject::tr("&Quit")
```

4) 다른 번역을 요구하는 같은 문자렬들의 식별

lupdate프로그람은 자동적으로 각 원천본문에 대하여 문맥을 제공한다. 이 문맥은 tr()호출을 포함하는 클라스의 이름이다. 이것은 거의 대부분의 경우에 충분하다. 그러나

때때로 번역기는 원천본문을 유일하게 확증하기 위하여 그 이상의 정보를 요구한다. 실례로 2개의 분리된 틀을 포함하는 대화칸을 들수 있는데 매개 틀에는 Enabled선택이 포함되여있으며 일부 언어들에서 둘사이에 번역이 차이나므로 매개의 식별이 요구된다. 이것은 tr()호출의 2인수형식을 사용하여 쉽게 달성한다. 례를 들면

```
rbc = new QRadioButton( tr("Enabled", "Color frame"), this );
```

과

```
rbh = new QRadioButton( tr("Enabled", "Hue frame"), this );
```

또한 Ctrl건 지름기호를 번역할수 있다.

```
file->insertItem( tr("E&xit"), qApp, SLOT(quit()), tr("Ctrl+Q", "Quit") );
```

tr()의 2인수형식이 Ctrl건 지름기호에 사용된다. 두번째 인수는 번역기가 지름기호에 의해 처리된 함수와 관련되여있다는 유일한 근거이다.

5) 항행정보에 의한 번역기의 방조

크고 복잡한 응용프로그람에서는 특별한 원천본문이 어디에 있는지 알기 힘들다. 이것은 문제의 본문에 도달하기 위한 항행단계를 서술하는 예약어 TRANSLATOR를 사용하는 주석문을 추가하여 해결할수 있다. 례를 들면

```
/*  TRANSLATOR FindDialog
    Choose Edit|Find from the menu bar or press Ctrl+F to pop up the Find dialog.
*/
```

이 설명문은 특히 창문부품 클라스에 쓸모있다.

6) C++이름공간의 복사

C++이름공간과 using namespace명령문이 lupdate를 혼란시킬수 있다. 그것은 MyClass가 MyNamespace이름공간에서 정의된다해도 MyClass::tr()를 MyNamespace::MyClass::tr()가 아니라 단지 그자체를 의미하는것으로 해석할수 있다. 그러므로 이 문자렬의 실행시번역은 실패한다.

MyClass::tr()를 사용하는 원천파일의 처음에 TRANSLATOR설명문을 놓음으로써 이 제한에 대하여 작업할수 있다.

```
/* TRANSLATOR MyNamespace::MyClass */
```

설명문뒤의 MyClass::tr()에 대한 모든 참고는 MyNamespace::MyClass::tr()를 의미하는것으로 해석된다..

7) QObject보조클라스의 밖에 있는 본문의 번역

QApplication::translate()의 사용

인용된 본문이 QObject보조클라스의 성원함수안에 없으면 적당한 클라스의 tr()함수 혹은 QApplication::translate()를 직접 사용한다.

```
void some_global_function( LoginWidget *logwid )
{
  QLabel *label = new QLabel(LoginWidget::tr("Password:"), logwid );
}
void same_global_function( LoginWidget *logwid )
{
  QLabel *label = new QLabel(qApp->translate("LoginWidget", "Password:"),
      logwid );
}
```

QT_TR_NOOP()와 QT_TRANSLATE_NOOP()의 사용

함수밖에서 번역가능한 본문을 완전히 가지고 있어야 한다면 2개의 마크로 즉 QT_TR_NOOP()와 QT_TRANSLATE_NOOP()를 사용할수 있다. 이 마크로들은 단지 lupdate에 의해 뽑아낼 본문을 선택할뿐이다. 마크로는 바로 본문(문맥없이)으로 전개한다.

QT_TR_NOOP()의 실례:

```
QString FriendlyConversation::greeting( int greet_type )
{
  static const char* greeting_strings[] = {
    QT_TR_NOOP( "Hello" ),
    QT_TR_NOOP( "Goodbye" )
  };
  return tr( greeting_strings[greet_type] );
}
```

QT_TRANSLATE_NOOP()의 실례:

```
static const char* greeting_strings[] = {
  QT_TRANSLATE_NOOP( "FriendlyConversation", "Hello" ),
  QT_TRANSLATE_NOOP( "FriendlyConversation", "Goodbye" )
};

QString FriendlyConversation::greeting( int greet_type )
{
  return tr( greeting_strings[greet_type] );
}
```

```
QString global_greeting( int greet_type )
{
   return qApp->translate( "FriendlyConversation",
                     greeting_strings[greet_type] );
}
```

2. 실례

3개의 실례를 제시한다. 첫 실례는 QTranslator객체의 창조를 보여준다. 또한 사용자가 볼수 있고 번역하려는 원천본문을 표식하는데 tr()함수를 사용하는 가장 간단한 방법을 보여준다. 둘째 실례는 현재 지역에 적용할수 있는 번역파일을 응용프로그람이 적재하게 하는 방법을 설명한다. 또한 2인수형식의 tr()를 사용하여 번역기에 대한 추가정보를 제공하는 방법을 보여준다. 셋째 실례는 등가한 원천본문들이 같은 문맥에서 발생할 때에도 그것들을 구별하는 방법을 설명한다. 또한 응용프로그람이 갱신될 때 번역도구들이 번역기의 작업을 최소화하도록 방조하는 방법을 론의한다.

1) 실례1: 번역의 적재와 사용



그림 2-6. 실례1(영문판)

·tt1.pro

```
TEMPLATE      = app
CONFIG       += qt warn_on
SOURCES       = main.cpp
TRANSLATIONS  = tt1_la.ts
```

·main.cpp

```
// Translation tutorial 1
#include <qapplication.h>
#include <qpushbutton.h>
#include <qtranslator.h>
int main( int argc, char **argv )
{
   QApplication app( argc, argv );
   QTranslator translator( 0 );
   translator.load( "tt1_la", "." );
   app.installTranslator( &translator );
```

```
    QPushButton hello( QPushButton::tr("Hello world!"), 0 );
    app.setMainWidget( &hello );
    hello.show();
    return app.exec();
}
```

이 실례는《Qt일반지식》2장 2절의 "hello-world"실례를 라틴문번역으로 다시 만든다. 그림 2-6은 영문판을 보여준다.

- 프로그람설명

```
#include <qtranslator.h>
```

이 행은 QTranslator클라스의 정의를 포함한다. 이 클라스의 객체들은 사용자가 볼수 있는 본문에 대해 번역을 제공한다.

```
QTranslator translator( 0 );
```

부모없는 QTranslator객체를 하나 생성한다.

```
translator.load( "tt1_la", "." );
```

프로그람에서 사용되는 원천본문에 대한 라틴번역을 포함하는 tt1_la.qm(.qm파일확장자는 암시적이다.)라는 파일을 적재하려고 한다. 파일이 발견되지 않으면 어떤 오유도 발생하지 않는다.

```
app.installTranslator( &translator );
```

tt1_la.qm으로부터 프로그람에 의하여 사용되는 번역풀(pool)에로 번역을 추가한다.

```
QPushButton hello( QPushButton::tr("Hello world!"), 0 );
```

"Hello world!"를 현시하는 누름단추를 하나 생성한다. tt1_la.qm을 발견하고 "Hello world!"에 대한 번역을 포함한다면 번역이 나타나고 그렇지 않으면 나타나지 않는다.

QObject를 계승하는 모든 클라스들은 tr()함수를 가지고있다. QObject클라스의 성원함수안에서 QPushButton::tr("Hello world!") 혹은 QObject::tr("Hello world!")대신에 간단히 tr("Hello world!")라고 쓴다.

- 영문으로 응용프로그람의 실행

번역파일tt1_la.qm를 만들지 않았으므로 응용프로그람을 실행할 때 원천본문이 표시된다.

- 라틴통보문파일의 생성

첫단계는 프로젝트를 위한 모든 원천파일들을 목록하는 프로젝트파일(tt1.pro)을 만드는것이다. 프로젝트파일은 qmake프로젝트파일 또는 심지어 보통의 makefile일수 있다. 다음과 같이

```
SOURCES       = main.cpp
```

```
TRANSLATIONS   = tt1_la.ts
```

를 포함하는 파일을 작성한다. TRANSLATIONS는 우리가 유지하려고 하는 통보문파일을 지정한다. 이 실례에서는 한조의 번역(즉 라틴)을 유지한다.

파일확장자는 .qm이 아니라 .ts이다. .ts번역원천형식은 응용프로그람을 개발할 때 사용하기 위하여 설계된다. 프로그람작성자 또는 개발관리자는 lupdate프로그람을 실행하여 실행원천코드에서 추출되는 원천본문을 가진 .ts파일을 생성하고 갱신한다. 번역기는 번역을 추가하고 편집하는 Qt Linguist를 사용하여 .ts파일을 읽고 갱신한다.

.ts형식은 직접 전자우편으로 보낼수 있고 판조종하기 쉽고 사람이 읽을수 있는 XML이다. 만일 이 파일을 수동적으로 편집한다면 XML를 위한 기정부호화가 Latin-1(ISO 8859-1)이 아니라 UTF-8이라는것을 알고있다. 'ø'(o에 사선이 있는 노르웨이어)와 같은 Latin-1문자로 입력하는 한가지 방법은 XML실체 "ø"를 사용하는것이다. 이것은 어떤 유니코드문자라도 작업한다.

번역이 끝나면 lrelease프로그람은 .ts파일을 .qm Qt통보문파일형식으로 바꾸는데 사용된다. .qm형식은 매우 빠른 탐색기능을 주기 위해 설계된 조밀한 2진형식이다. lupdate와 lrelease는 둘다 프로젝트의 전체 원천와 머리부파일(프로젝트파일의 HEADERS와 SOURCES행에 지정된 파일)들을 읽어들이고 tr()함수호출에 나타나는 문자렬들을 뽑아낸다.

lupdate는 원천코드와 동조하여 유지하도록 통보문파일(이 경우에 tt1_la.ts)들을 생성하고 갱신하는데 사용된다. 언제든지 lupdate를 실행하는것은 안전하다. lupdate는 정보도 조금도 삭제하지 않는다. 례를 들면 그것을 makefile에 넣어서 원천이 변경될 때마다 .ts파일을 갱신할수 있다.

다음과 같이 lupdate를 실행할수 있다.

```
lupdate -verbose tt1.pro
```

(-verbose선택은 lupdate에게 그것이 무엇을 하고 있는가를 설명하는 통보문를 표시하게 한다.) 현재 등록부에는 다음의 내용을 포함하는 파일 tt1_la.ts가 있어야 한다.

```
<!DOCTYPE TS><TS>
<context>
   <name>QPushButton</name>
   <message>
      <source>Hello world!</source>
      <translation type="unfinished"></translation>
   </message>
</context>
</TS>
```

이 파일은 도구(lupdate, Qt Linguist, lrelease)를 사용하여 읽어들이고 갱신하므로 파일형식을 해석할 필요가 없다.

- Qt Linguist에 의하여 라틴문으로 번역

Qt Linguist를 사용하여 번역을 제공한다. 그렇지만 XML 또는 평본문편집기를 사용하여 번역을 .ts파일에 입력할수 있다.

Qt Linguist를 기동하려면 다음과 같이 입력한다.

linguist tt1_la.ts

지금 왼쪽웃구석에서 본문 "QPushButton"가 있다. 그것을 두번 련속 누르고 "Hello world!"우를 누르고 Translation구획(창문의 오른쪽 중간)에 "Orbis, te saluto!"라고 입력한다. 느낌표!를 잊지 말아야 한다.

Done검사칸을 누르고 차림표띠에서 File|Save를 선택한다. .ts파일은 더이상

<translation type='unfinished'></translation>

을 포함하지 않지만 그대신

<translation>Orbis, te saluto!</translation>

를 가진다.

- 라틴문으로 응용프로그람의 실행

라틴어로 실행하는 응용프로그람을 보기 위하여 .ts파일로부터 .qm파일을 생성해야 한다. .qm파일은 Qt Linguist로부터(하나의 .ts파일을 위해) 생성하거나 혹은 프로젝트파일에 목록되는 매개 .ts파일에 대하여 하나의 .qm파일을 생성하는 지령행프로그람 lrelease를 사용하여 생성할수 있다. Qt Linguist의 차림표띠에서 File|Release를 선택하고 올리펼쳐지는 파일보존대화칸에서 Save를 눌러서 tt1_la.ts로부터 tt1_la.qm을 생성한다. 이제 tt1실레프로그람을 다시 실행한다. 이번에 단추에는 "Orbis, te saluto!"로 된다.



그림 2-7. 실레1(라틴어판)

2) 실레2: 2개이상 언어의 사용

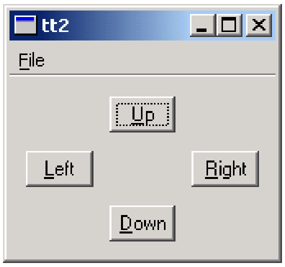


그림 2-8. 실레2(영문판)

·tt2.pro

```
TEMPLATE       = app
CONFIG         += qt warn_on
HEADERS        = arrowpad.h \
                 mainwindow.h
SOURCES        = arrowpad.cpp \
                 main.cpp \
                 mainwindow.cpp
TRANSLATIONS   = tt2_fr.ts \
                 tt2_nl.ts
```

이 실례는 좀 더 복잡하며 중요한 Qt Linguist개념문맥을 소개한다.

- arrowpad.h는 사용자정의창문부품 ArrowPad의 정의를 포함한다.
- arrowpad.cpp는 ArrowPad의 실현을 포함한다.
- mainwindow.h는 QMainWindow의 보조클라스 MainWindow의 정의를 포함한다.
- mainwindow.cpp는 MainWindow의 실현을 포함한다.
- main.cpp는 main()를 포함한다.

2개의 번역 프랑스어와 네테를란드어를 사용하지만 응용프로그람에서 사용가능한 번역의 수에 대한 어떤 효과적인 제한도 없다. tt2.pro의 련관된 행들은 다음과 같다.

```
HEADERS        = arrowpad.h \
                 mainwindow.h
SOURCES        = arrowpad.cpp \
                 main.cpp \
                 mainwindow.cpp
TRANSLATIONS   = tt2_fr.ts \
                 tt2_nl.ts
```

lupdate을 실행하면 2개의 등가한 통보문파일 tt2_fr.ts와 tt2_nl.ts를 만들어낸다. 이 파일들은 tr()호출에 의해 번역하려고 표식한 모든 원천본문들과 그 문맥들을 포함한다.

- 프로그람설명

arrowpad.h에서는 QWidget의 파생클라스인 ArrowPad클라스를 정의한다. 그림 2-8에서 4개의 단추를 가지는 중심창문부품이 ArrowPad이다.

```
class ArrowPad : public QGrid
```

lupdate를 실행하면 원천본문만 꺼내는것이 아니라 그것들을 문맥들로 묶는다. 문맥은 원천본문이 나타나는 클라스의 이름이다. 이리하여 이 실례에서 "ArrowPad"는 문맥이고 그것은 ArrowPad클라스안에 있는 본문들의 문맥이다. Q_OBJECT마크로는

ArrowPad에서 tr(x)를 다음과 같이 정의한다.

```
qApp->translate( "ArrowPad", x )
```

매개 원천본문이 나타나는 클라스를 알면 Qt Linguist가 론리적으로 모두 련관된 본문들을 묶을수 있게 한다. 실례로 대화칸의 모든 본문은 대화칸의 클라스이름의 문맥을 가지며 모두 표시된다. 이것은 본문이 나타나는 문맥이 번역방법에 영향을 줄수 있으므로 번역기에 유용한 정보를 제공한다. 일부 번역에서 지름건들은 변경할 필요가 있으며 모두 묶어진 특정한 문맥(클라스)안의 모든 원천본문을 취하여 번역기가 총돌을 일으킴이 없이 지름건의 변경을 간단히 수행하게 한다.

arrowpad.cpp에서는 ArrowPad클라스를 실현한다.

```
(void) new QPushButton( tr("&Up"), this );
```

표식자가 사용자가 볼수 있는 본문이므로 매개 단추의 표식에 대하여 ArrowPad::tr()를 호출한다.

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
```

그림 2-8에서 전체 창문은 하나의 MainWindow이다. 이것은 mainwindow.h머리부파일에 정의된다. 여기서도 Q_OBJECT를 사용하므로 MainWindow는 Qt Linguist안의 문맥으로 된다.

MainWindow, mainwindow.cpp의 실현에서는 ArrowPad클라스의 실례를 창조한다.

```
ArrowPad *ap = new ArrowPad( this, "arrow pad" );
```

또한 MainWindow::tr()를 두번 호출한다. 한번은 차림표항목에 대하여 또한번은 지름건에 대하여 호출한다.

```
file->insertItem( tr("E&xit"), qApp, SLOT(quit()), tr("Ctrl+Q", "Quit") );
```

tr()의 사용은 다른 언어들에서 서로 다른 건들을 유지한다. "Ctrl+Q"는 영문에서 Quit의 좋은 선택이지만 네테를란드번역기는 "Ctrl+A" (Afsluiten)를 사용할것을 요구하며 도이칠란드번역기는 "Strg+E" (Beenden)를 요구한다. Ctrl지름건에 tr()를 사용할때 둘째 인수가 지름건이 수행하는 기능을 서술하는 2인수형식을 사용해야 한다.

main()함수는 보통과 같이 main.cpp에서 정의된다.

```
QTranslator translator( 0 );
translator.load( QString("tt2_") + QTextCodec::locale(), "." );
app.installTranslator( &translator );
```

현재 지역에 따라서 사용하려는 번역을 선택한다. 가령 QTextCodec::locale()은 LANG환경변수의 설정에 의해 영향을 받을수 있다. .qm통보문파일들(과 .ts파일들)을 위한 지역을 결합하는 명명관례의 사용이 지역에 따르는 번역파일의 선택을 실현하기 쉽게

한다.

선택된 지역에 대한 .qm통보문파일이 없으면 원래 원천본문이 사용되고 오유가 일어나지 않는다.

- 프랑스어와 네테를란드어로 번역

우선 실례응용프로그람을 프랑스어로 번역하자. tt2_fr.ts를 가지고 Qt Linguist를 기동한다. 두개의 문맥("ArrowPad"와 "MainWindow")안에 그룹화된 7개의 원천본문들("&Up", "&Left" 등)을 얻어야 한다.

이제 다음의 번역을 입력한다.

- ArrowPad

&Up - &Haut

&Left - &Gauche

&Right - &Droite

&Down - &Bas

- MainWindow

E&xit - &Quitter

Ctrl+Q - Ctrl+Q

&File - &Fichier

매번 번역을 입력한 다음 Alt+D(Done & Next단추를 찰칵)를 누르는것이 가장 빠르다. 이것은 번역을 수행한것으로 표식하고 다음 원천본문으로 이동한다.

파일을 보관하고 tt2_nl.ts:를 가지고 네테를란드어에 대하여 같은 조작을 수행한다.

- ArrowPad

&Up - &Boven

&Left - &Links

&Right - &Rechts

&Down - &Onder

- MainWindow

E&xit - &Afsluiten

Ctrl+Q - Ctrl+A

File - &Bestand

tt1_fr.ts과 tt1_nl.ts번역원천파일들을 .qm파일들로 변환하여야 한다. 이전에 수행한것처럼 Qt Linguist를 사용할수 있으나 지령행도구 lrelease를 사용하여 Qt Linguist로부터 매개 파일을 개별적으로 적재하고 File|Release함이 없이 응용프로그람을 위한 모든 .qm파일들을 창조한다는것을 담보한다.

실천에서는 응용프로그람의 makefile에 lupdate와 lrelease의 호출을 포함하여 마

지막 번역들이 사용되도록 담보한다.

다음과 같이 입력한다.

```
lrelease tt2.pro
```

이것은 tt2_fr.qm와 tt2_nl.qm를 둘다 창조한다. LANG환경변수를 fr로 설정한다. Unix에서는 다음의 두 지령들중 하나가 작업해야 한다.

```
export LANG=fr
setenv LANG fr
```

Windows에서는 autoexec.bat를 수정하거나 다음과 같이 실행한다.

```
set LANG=fr
```

이제 프로그람을 실행하면 프랑스판을 볼수 있다(그림 2-9).

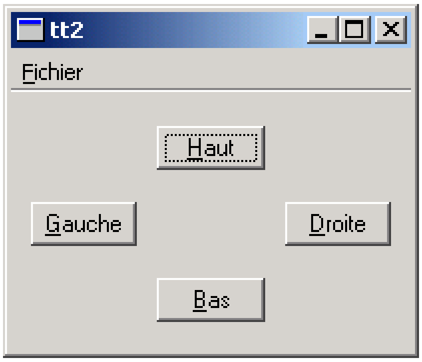


그림 2-9. 실례2(프랑스판)

LANG=nl라고 설정하고 네데를란드어로 같은것을 수행한다. 이제는 네데를란드판이 나타난다.

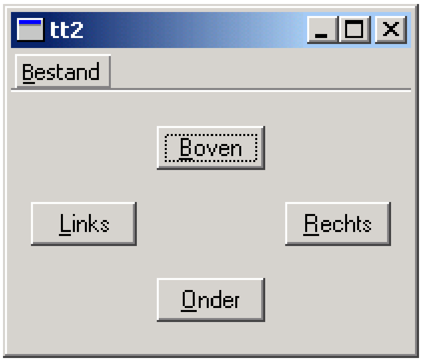


그림 2-10. 실례2(네데를란드판)

- 련습문제

Qt Linguist에서 수행하지 않은 번역들중 하나를 표식한다. 즉 "done"검사칸을 해제한다. 그리고 lupdate를, 그다음 lrelease, 그다음 실례를 실행한다. 이러한 변경이 어떤 효과를 가지는가?

LANG=fr_CA (프랑스 카나다)로 설정하고 실례프로그람을 다시 실행한다. 결과가 LANG=fr인 경우와 같은 원인을 설명하시오.

&Bestand와 &Boven사이의 충돌을 없애기 위하여 네테를란드번역에서 지름건들중 하나를 변경한다.

3) 실례3: 등가한 문자렬들의 애매한 점을 없앤다

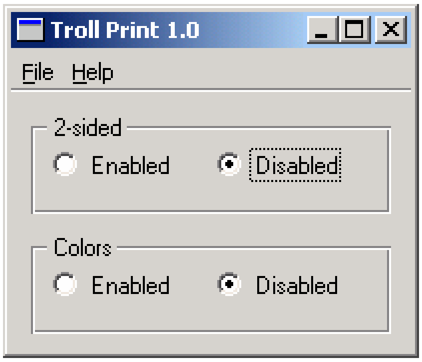


그림 2-11. 실례3, "Troll Print 1.0" 영문판

·tt3.pro

```
TEMPLATE      = app
CONFIG       += qt warn_on
HEADERS       = mainwindow.h \
                printpanel.h
SOURCES       = main.cpp \
                mainwindow.cpp \
                printpanel.cpp
TRANSLATIONS  = tt3_pt.ts
```

이 실례에 대하여 뽀르뚜갈어번역문들을 포함하는 번역파일 tt3_pt.ts을 포함하였다.

같은 응용프로그람의 2개의 출하판 Troll Print 1.0과 1.1을 고찰한다. 첫 출하판용으로 창조한 번역문들을 다음의 출하판에서 재리용하는 방법을 학습한다. (이 실례에서는 일부 원천파일들을 편집할 필요가 있다. 새로운 일시등록부에 모든 파일들을 복사해두고 거기서 작업하는것이 아마 제일 좋을것이다.)

Troll Print는 사용자가 인쇄기환경을 선택하게 하는 장난감과 같은 실례프로그람이다. 두개의 판 즉 영문판과 뽀르뚜갈문판이 있다.

판1.0은 다음의 파일들로 이루어진다.

- printpanel.h는 PrintPanel의 정의를 포함한다.
- printpanel.cpp는 PrintPanel의 실현을 포함한다.
- mainwindow.h는 MainWindow의 정의를 포함한다.

- mainwindow.cpp는 MainWindow의 실현을 포함한다.
- main.cpp는 main()을 포함한다.
- tt3.pro는 qmake 프로젝트파일이다.
- tt3_pt.ts는 뽀르뚜갈어통보파일이다.

- 프로그람설명

PrintPanel는 printpanel.h에서 정의된다.

```
class PrintPanel : public QVBox
{
    Q_OBJECT
```

PrintPanel는 QWidget이다. tr()가 적당히 작업하려면 Q_OBJECT마크로를 요구한다.

실현파일은 printpanel.cpp이다.

```
/*
    QLabel *lab = new QLabel( tr("<b>TROLL PRINT</b>"), this );
    lab->setAlignment( AlignCenter );
*/
```

Troll Print 1.0에서 일부 코드는 설명문으로 하였는데 후에 Troll Print 1.1에서 그것을 실행문으로 고친다.

```
    QHButtonGroup *twoSided = new QHButtonGroup( this );
    twoSided->setTitle( tr("2-sided") );
    but = new QRadioButton( tr("Enabled"), twoSided );
    but = new QRadioButton( tr("Disabled"), twoSided );
    but->toggle();
    QHButtonGroup *colors = new QHButtonGroup( this );
    colors->setTitle( tr("Colors") );
    but = new QRadioButton( tr("Enabled"), colors );
    but = new QRadioButton( tr("Disabled"), colors );
    but->toggle();
```

PrintPanel에서 tr("Enabled")와 tr("Disabled")가 두번 출현한다. "Enabled"와 "Disabled"들이 둘다 같은 문맥에서 나타나므로 Qt Linguist는 매개의 하나의 출현만 현시하며 현시하지 않는 중복되는것들에 대해서는 같은 번역을 사용한다. 이것은 뽀르뚜갈어와 같은 일부 언어들에서 시간을 절약하는데 아주 쓸모있으며 한편 두번째 출현은 따로따로의 번역을 요구한다. Qt Linguist가 따로따로 번역하는데 맞게 모든 출현을 현시하도록 하는 방법을 간단히 고찰하자.

MainWindow의 머리부파일 mainwindow.h은 새로운것을 포함하지 않는다. 실현파일

mainwindow.cpp에는 번역에 대하여 표식되여야 하는 사용자가 볼수 있는 원천본문들이 있다.

```
setCaption( tr("Troll Print 1.0") );
```

창문의 제목을 번역해야 한다.

```
file->insertItem( tr("E&xit"), qApp, SLOT(quit()), tr("Ctrl+Q", "Quit") );
QPopupMenu *help = new QPopupMenu( this );
help->insertItem( tr("&About"), this, SLOT(about()), Key_F1 );
help->insertItem( tr("About &Qt"), this, SLOT(aboutQt()) );
menuBar()->insertItem( tr("&File"), file );
menuBar()->insertSeparator();
menuBar()->insertItem( tr("&Help"), help );
```

또한 차림표항목들도 번역해야 한다. 2인수형식의 tr()가 지름건 "Ctrl+Q"에 사용되며 둘째 인수는 그 지름건이 수행하는 기능이 무엇인가를 번역기가 가리키는 유일한 실마리이다.

```
QTranslator translator( 0 );
translator.load( QString("tt3_") + QTextCodec::locale(), "." );
app.installTranslator( &translator );
```

main.cpp의 main()함수는 실례2에서와 같다. 특히 이 함수는 현재 지역에 기초하여 번역파일을 선택한다.

- 영문과 뽀르뚜갈어로 Troll Print 1.0의 실행

제공되여있는 tt3_pt.ts파일의 번역을 리용한다.

LANG환경변수를 pt로 설정한 다음 tt3를 실행한다. 여전히 그림 2-11에서 보여준바와 같이 영문이 표시되여야 한다. 이제 lrelease 실례로 lrelease tt3.pro를 실행하고 다시 실례를 실행한다. 그러면 뽀르뚜갈어판(Troll Imprimir 1.0)이 표시되여야 한다.

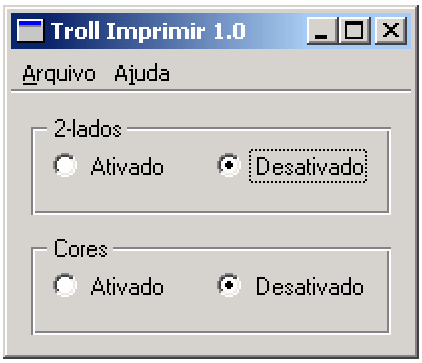


그림 2-12. 실례3, "Troll Imprimir 1.0", (나쁜) 뽀르뚜갈판

번역은 정확히 표시되였으나 사실상 나쁘다. 좋은 뽀르뚜갈문에서 "Enabled"의 두번째 출현은 "Ativado"가 아니라 "Ativadas"로 되여야 하고 "Disabled"의 둘째번역의 마감에도 마찬가지로 변경되여야 한다.

Qt Linguist를 리용하여 tt3_pt.ts을 열면 번역원천파일안에 "Enabled"과 "Disabled"의 바로 한번 출현하는것을 볼수 있다. 그렇지만 원천코드안에 각각 두개씩 있다. 이것은 Qt Linguist가 중복되는 원천본문에는 같은 번역문을 사용함으로써 번역기의 작업을 최소화하려고 하기때문이다. 하나의 동일한 번역이 틀리는 경우에 프로그람작성자는 중복되는 출현에서 모호한 점을 없애야 한다. 이것은 2인수형식의 tr()를 사용하여 쉽게 이루어진다.

번역기의《문맥》이 사실상 변경되여야 할 본문이 나타나는 클라스의 이름이므로 어느 파일이 변경되여야 하는가를 쉽게 결정할수 있다. 이 경우에 파일은 printpanel.cpp이고 거기에는 변경해야 할 행이 4개 있다. 라지오단추들의 첫 쌍에 대한 적당한 tr()호출들에 둘째 인수 "two-sided"를 추가한다.

```
but = new QRadioButton( tr("Enabled", "two-sided"), twoSided );
but = new QRadioButton( tr("Disabled", "two-sided"), twoSided );
```

그리고 라지오단추들의 둘째 쌍에 대한 적당한 tr() 호출들에 둘째 인수 "colors"를 추가한다.

```
but = new QRadioButton( tr("Enabled", "colors"), colors );
but = new QRadioButton( tr("Disabled", "colors"), colors );
```

이제는 lupdate을 실행하고 Qt Linguist에서 tt3_pt.ts을 연다. 그러면 두가지 변경을 볼수 있다.

첫째로, 지금 번역원천파일은 3개의 "Enabled", "Disabled"쌍을 포함한다. 첫째쌍은 "(obs.)"로 표시하여 그것들이 낡은것이라는것을 보여준다. 그것은 이 본문들이 2인수를 가진 새로운 호출들로 교체된 tr()호출에 나타났기때문이다. 둘째 쌍은 주석으로서 "two-sided"를, 셋째 쌍은 "colors"를 가진다. 주석은 Qt Linguist의 원천본문과 주석구역에 표시된다.

둘째로, 번역본문 "Ativado"와 "Desativado"는 새로운 "Enabled"과 "Disabled" 본문들용의 번역과 같이 자동적으로 사용되여 번역기의 작업을 최소화한다. 물론 이 경우에 매개 단어가 두번째로 출현하는 경우에 정확하지 않지만 좋은 출발점을 제공한다.

두번째의 "Ativado"를 "Ativadas"로, 두번째의 "Desativado"를 "Desativadas"로 변경한 다음 보관하고 완료한다. lrelease를 실행하여 갱신된 2진파일 tt3_pt.qm을 얻고 Troll Print (혹은 Troll Imprimir)를 실행한다.

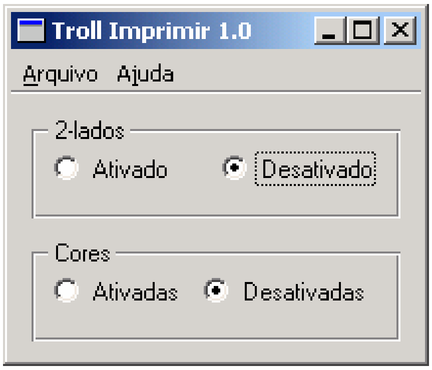


그림 2-13. 실례3, "Troll Imprimir 1.0", (좋은) 뽀르뚜갈판

Qt Linguist에서 《주석》이라고 부르는 tr()호출에로의 제2인수는 같은 문맥(클라스)에서 발생하는 동일한 원천본문들사이를 구별한다. 그것들은 또한 번역기에 실마리를 주기 위하여 다른 경우들에 사용할수 있으며 Ctrl지름건의 경우에는 번역기에로의 지름건에 의하여 수행된 기능을 전달하는 유일한 수단이다.

번역기를 방조하는 추가적인 방법은 번역해야 할 원천본문들을 포함하는 응용프로그람의 특정부분에로 항행하는 방법에 대한 정보를 제공하는것이다. 이것은 번역문이 나타나는 문맥을 알수 있게 방조하며 또한 번역문들을 찾고 시험하는데 도움을 준다. 이것은 원천코드에서 TRANSLATOR주석을 히용함으로써 이루어진다.

```
/* TRANSLATOR MainWindow
     In this application the whole application is a MainWindow.
     Choose Help|About from the menu bar to see some text
     belonging to MainWindow.
*/
```

일부 원천파일들에 특히는 대화칸클라스들에 주석을 추가하여 대화칸에 도달하는데 필요한 항행을 서술한다. 또한 실례파일들에 주석을 추가할수 있다. 실례로 mainwindow.cpp와 printpanel.cpp가 적당한 파일이다. lupdate을 실행한 다음 Qt Linguist를 기동하고 tt3_pt.ts를 적재한다. 원천본문을 열람할 때 원천본문 및 주석구역에서 주석을 볼수 있다.

때때로 특히 대규모프로그람들에서 번역기가 그 번역문들을 찾고 그것들이 정확한가를 찾기는 힘들다. 좋은 항행정보를 제공하는 주석은 시간을 절약할수 있다.

```
/* TRANSLATOR ZClientErrorDialog
    Choose Client|Edit to reach the Client Edit dialog, then choose
    Client Specification from the drop down list at the top and pick
    client Bartel Leendert van der Waerden. Now check the Profile
    checkbox and then click the Start Processing button. You should
```

now see a pop up window with the text "Error: Name too long!".
This window is a ZClientErrorDialog.
*/

- Troll Print 1.1

이제는 Troll Print의 출하판 1.1을 준비하자. 자기가 좋아하는 본문편집기를 기동하고 다음과 같은 걸음들을 수행한다.

• printpanel.cpp에서 본문 "TROLL PRINT"를 가지는 QLabel을 하나 창조하는 두개 행의 주석을 해제한다.

• 단어배치: printpanel.cpp에서 "2-sided"를 "Two-sided"로 교체한다.

• mainwindow.cpp에서 모든 "1.0"을 "1.1"로 교체한다.

• mainwindow.cpp에서 저작권년도를 1999-2000로 교체한다.

(물론 판번호와 저작권년도는 실제 응용프로그람에서 const 혹은 #define일수 있다.)

완료하면 lupdate을 실행하고 Qt Linguist에서 tt3_pt.ts을 연다. 다음의 항목들은 특별한 흥미를 가진다.

• MainWindow

Troll Print 1.0 - 낡은것을 의미하는 "(obs.)"로 표식된다.

About Troll Print 1.0 -낡은것을 의미하는 "(obs.)"로 표식된다.

Troll Print 1.0. Copyright 1999 Macroshaft, Inc. -낡은것을 의미하는 "(obs.)"로 표식된다.

Troll Print 1.1 - "Troll Imprimir 1.1"로 자동번역된다.

About Troll Print 1.1 - "Troll Imprimir 1.1"로 자동번역된다.

Troll Print 1.1. Copyright 1999-2000 Macroshaft, Inc. - "Troll Imprimir 1.1. Copyright 1999-2000 Macroshaft, Inc." 로 자동번역된다.

• PrintPanel

2-sided -낡은것을 의미하는 "(obs.)"로 표식된다.

TROLL PRINT - 표식되지 않는다. 즉 번역되지 않는다.

Two-sided - 표식되지 않는다. 즉 번역되지 않는다.

lupdate가 수정을 더 간단히 하기 위하여 배경에서 힘들게 작업한다는것을 알아두시오.

MainWindow안의 번역문들로 가서 그것들을 Done으로 표식한다. "TROLL PRINT"를 "TROLL IMPRIMIR"로 번역한다. "Two-sided"를 번역하고있을 때 Guess Again단추를 찰칵하여 "Two-sided"를 번역하지만 "2"를 "Dois"로 번역한다.

보관 및 완료한 다음 lrelease를 실행한다. 뽀르뚜갈어판이 다음과 같이 표시된다.



그림 2-14. 실례3, "Troll Imprimir 1.1", 뽀르뚜갈판

Ajuda|Sobre, (Help|About)를 선택하여 판정보대화칸을 본다.

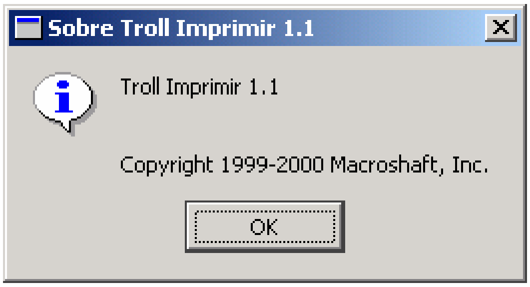


그림 2-15. 실례3, About box, 뽀르뚜갈판

Ajuda|Sobre Qt, (Help|About Qt)을 선택한다면 영문대화칸을 얻는다. 아차! Qt 자체로 번역할 필요가 있다(《Qt일반지식》 7장 1절을 참고).

그러면 LANG=en로 설정하여 원래의 영문판을 얻는다.

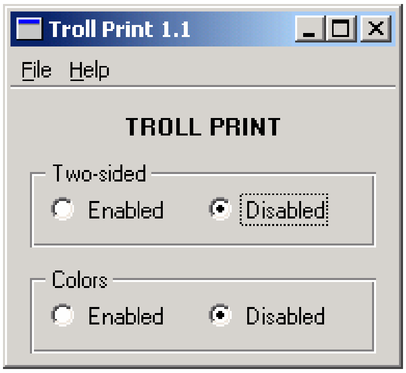


그림 2-16. 실례3, "Troll Print 1.1", 영문판

실례들에서 번역용의 Qt응용프로그람을 준비하기 위하여 알아두어야 할 모든것을 설명하였다.

프로젝트의 시작에서 프로젝트파일에 사용하려는 번역원천파일들을 추가하고 makefile에 lupdate와 lrelease에로의 호출을 추가한다.

프로젝트에서 프로그람작성자가 수행하는 모든것은 tr()호출에서 사용자에게 보이는 본문의 일부분을 가리운다. 또한 같은 문맥에서 같은 본문이 두개의 다른 형식으로 번역되는 경우에 번역기가 질문할 때 2인수형식의 Ctrl지름건을 사용해야 한다. 또한 프로그람작성자는 TRANSLATION주석을 포함하여 번역기가 응용프로그람을 항행하는데 도움을 준다.

제3장. Qt Assistant

이 장에서는 직결문서를 표시하는 도구인 Qt Assistant를 소개한다. 또한 Qt Assistant를 사용하여 혹은 웨브열람기로 호출할수 있는 Qt참고문서를 소개한다.

제1절. Qt참고문서에 대한 소개

Qt서고용문서는 개발자들에 의하여 .cpp파일들에 직접 씌여진다. 문서팀은 문서를 교정하여 그것이 정확하고 사용할수 있게 높은 질을 담보하도록 한다. 또한 문서팀은 클라스가 사용하는 개념, 클라스가 제공하는 함수와 속성들을 소개하는 더 큰 본문을 작성한다.

문서는 그 내부보다도 API에 초점을 둔다. API가 매개 판에서 일관성있고 호환성있게 유지하도록 큰 품을 들이지만 내부를 상당히 변경하여 성능을 개선하고 기능을 강화할수 있다.

Qt참고문서는 약 1,500개의 HTML페지(2,500이상의 인쇄페지)로 이루어진다. 그 대부분의 페지는 Qt클라스들을 서술한다. 개발자들은 사고방식과 작업방법이 서로 다르므로 문서모임을 항행하는 몇가지 수법을 제공한다.

- 모든 클라스페지는 Qt의 공개API안의 각 클라스를 서술하며 수백개의 클라스들로 이루어진다.
- 기본클라스페지는 제일 흔히 사용하는 클라스들을 서술하며 모든 클라스목록보다 훨씬 더 간단히 더 취급하기 쉬운 목록을 제공한다.
- 클라스그룹페지는 매개 그룹이 관련된 클라스들의 목록으로 이루어지는 그룹목록을 제시한다. 실례로 고급한 창문부품목록을 들수 있다.
- 계승도페지는 Qt클라스들의 계층도로 클라스들의 목록을 제시한다.
- 모든 함수페지는 Qt클라스에 의해 제공된 모든 함수들을 목록화한다. 매개 함수는 그것이 있는 클라스들과 련결된다.

Qt문서의 탐색에는 문제가 없으며 포괄적인 교차참고를 찾을수 있다. 실례코드의 단편에 선택가능한 련결이 포함되여도 실례로 코드실례에서 클라스선언을 우연히 만난다면 클라스이름은 그 클라스의 문서에 대한 선택가능한 련결로 될수 있다.

클라스문서와 함께 일부 Qt모듈은 대량적으로 서술되여있다. Qt서고의 각종 견해를 서술하는 개괄문서가 많으며 이것들은 모두 참고문서의 홈페지로부터 련결되여있다. 또한 Qt배포물의 examples보조등록부에는 두개의 련습지도서와 많은 량의 실례프로그람들이 있다.

제2절. Qt Assistant사용법에 대한 소개

Windows에서 Qt Assistant는 Qt차림표에서 차림표선택으로 사용할수 있다. Unix에서는 xterm으로부터 assistant를 실행한다.

Qt Assistant를 기동할 때에는 차림표띠와 도구띠가 있는 표준기본창문형식의 응용프로그람이 주어진다. 그 아래 왼쪽측면에 있는것이 가름띠(Sidebar)라고 부르는 항행창문이고 오른쪽에 대부분의 공간을 차지하는것이 문서창문이다. 기정으로 Qt참고문서의 홈페지는 문서창문에 표시된다.

Qt Assistant는 웨브열람기와 비슷한 방법으로 작업한다. 밑줄있는 본문(교차참고를 의미한다)을 선택하면 문서창문이 련관된 페지를 제시한다. 특별히 흥미있는 페지들에 서표를 만들수 있으며 Previous과 Next도구띠단추를 찰칵하여 방문하려는 페지들로 이행할수 있다.

Qt Assistant가 Qt문서모임을 항행하기 위하여 웨브열람기처럼 쓰이지만 Qt Assistant는 웨브열람기가 제공하지 못하는 강력한 항행수단을 제공한다. Qt Assistant는 문서모임의 모든 페지들을 색인화하는데 지능알고리듬을 리용함으로써 특별한 단어와 구들을 탐색할수 있다.

색인검색을 수행하려면 Sidebar의 Index타브를 선택한다. (또는 Ctrl+I를 누른다.) Look For행편집칸에 단어를 입력한다. 실례로 homedirpath를 들수 있다. 입력시에 단어들이 검색되고 행편집칸아래의 목록에 강조표시된다. 강조표시된 본문이 검색하려는것과 일치하면 그것을 두번 련속 찰칵한다.(또는 Enter를 누른다.) 이때 문서창문에는 련관된 페지가 현시된다. 드문히 Qt Assistant가 일치하는것을 검색하기전에 전체단어를 입력해야 한다. 일부 단어들에 대해서는 련관된 페지가 하나이상 있을수 있다.

Qt Assistant는 또한 문서에서 특정한 단어를 찾기 위한 완전본문검색을 제공한다. 검색하고있는 단어의 출현률이 제일 높은 문서를 먼저 표시하며 문서에서 단어가 출현할때마다 강조표시된다.

Qt Assistant는 문서집합인 프로파일을 창조하여 전용화할수 있다. 프로파일은 자기가 사용하기 위하여 혹은 배포하는 응용프로그람용으로 창조할수 있다. 프로파일을 리용하여 요구하는 문서를 선택하고 응용프로그람의 말단사용자가 보게 할수 있다.

제3절. Qt Assistant의 세부

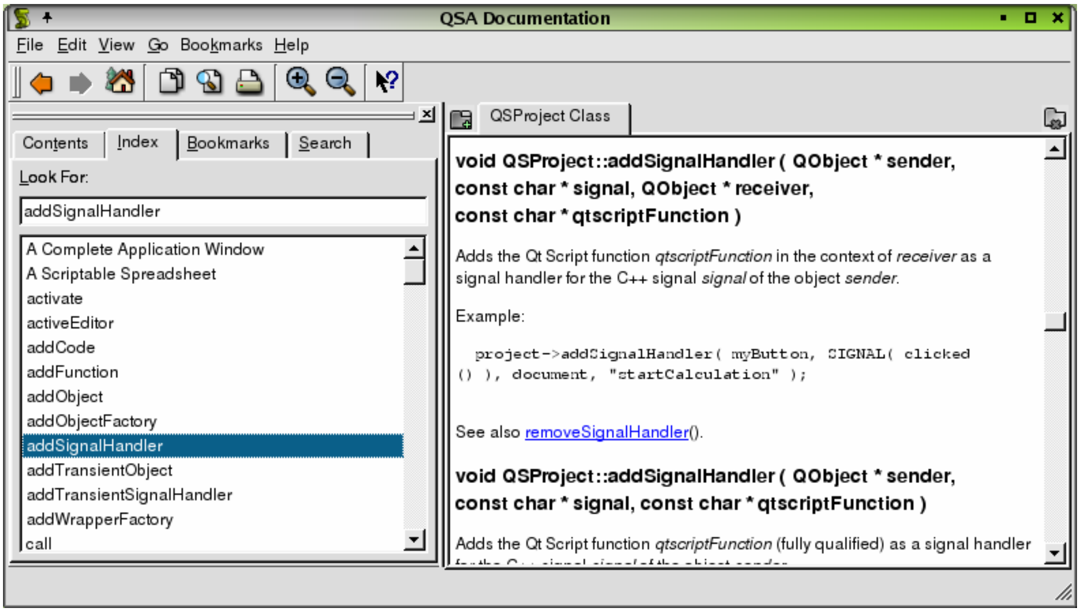


그림 3-1. Qt Assistant

1. 가름띠(Sidebar)

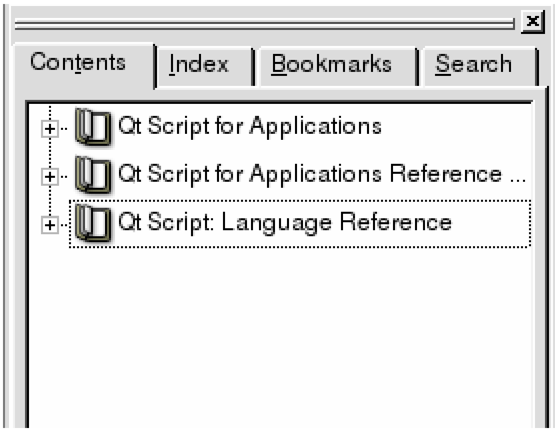


그림 3-2. 가름띠

가름띠는 문서를 항행하는 4가지 방법을 제공한다.

① Contents타브는 유효문서모임의 나무보기를 표시한다. 항목을 하나 누르면 그 문서가 문서창문에 나타난다. 항목을 두번 련속 누르거나 항목왼쪽의 +기호를 선택하면 항목의 보조항목들이 나타난다. 보조항목을 하나 누르면 문서창문에 그 페지가 나타난다. 항목왼쪽의 -부호를 선택하여 그 보조항목을 숨긴다.

② Index타브는 열쇠단어 혹은 구들을 검색하는데 리용된다(2절을 참고).

③ Bookmarks타브는 자기가 만든 서표들을 표시한다. 서표를 하나 두번 련속 누르면 문서창문에 그 페지가 나타난다. Bookmarks타브는 아래에 New Bookmark단추와 Delete Bookmark단추를 가지고있다. New Bookmark를 선택하여 문서창문에 보여주고있는 페지에 서표를 추가한다. 목록에서 서표를 하나 선택하고 Delete Bookmark를 눌러 강조표시된 서표를 삭제한다.

④ Search타브는 모든 문서의 완전본문검색을 제공한다(4절 참고).

문서창문이 될수록 더 많은 공간을 차지하게 하려면 가름띠를 간단히 숨기거나 표시할수 있다. 가름띠가 표시되여있으면 Ctrl+T, Ctrl+I를 누르고 숨기려면 Ctrl+B 혹은 Ctrl+S를 누른다. 가름띠가 숨겨져있으면 Ctrl+T를 눌러서 Contents타브에 표시하거나 또는 Ctrl+I를 눌러서 Index타브에 그것을 표시하거나(Look For행편집칸에 초점이 놓인다) 혹은 Ctrl+B를 눌러서 Bookmarks타브에 표시하고 또는 Ctrl+S를 눌러서 (완전본문) Search타브에 표시할수 있다.

가름띠는 류동가능창문이므로 Qt Assistant창문의 상하좌우로 끌고가거나 Qt Assistant밖에 끌고가서 그것이 떠오르게 할수 있다.

2. 문서창문

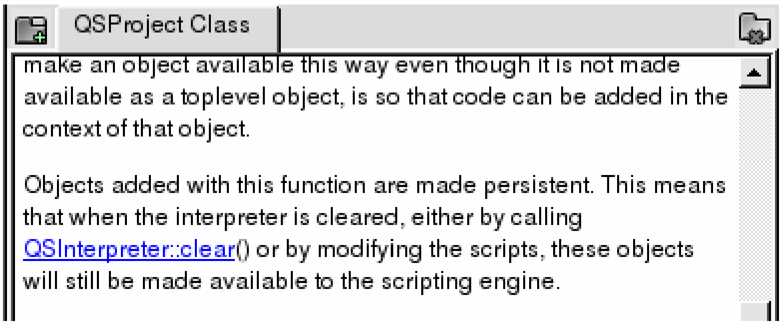


그림 3-3. 문서창문

문서창문은 표시하는 각 문서에 대하여 타브작성을 허용함으로써 문서표시기능을 제공한다. Add Tab단추를 누르면 새 타브가 타브의 제목으로서 페지이름과 함께 나타난다. 이것은 여러개의 각이한 문서들과 작업하고있을 때 페지들사이를 절환할수 있게 한다. 문서창문의 오른쪽에 있는 Close Tab단추를 찰칵하여 타브를 삭제할수 있다.

3. 도구띠



그림 3-4. 도구띠

도구띠는 가장 일반적으로 사용하는 작용들을 고속으로 호출할수 있게 한다.

• Previous는 이전 페지를 펼치게 한다. 차림표선택은 Go|Previous이고 지름건은 Alt+Left Arrow이다.

• Next는 다음 페지를 펼치게 한다. 차림표선택은 Go|Next이고 지름건은 Alt+Right Arrow이다.

• Home은 홈페지(보통 Qt참고문서의 홈페지)를 펼치게 한다. 차림표선택은 Go|Home, 지름건은 Ctrl+Home이다.

• Copy는 선택된 본문을 오려둠판에 복사한다. 차림표선택은 Edit|Copy이고 지름건은 Ctrl+C이다.

• Find in Text는 Find Text대화칸을 펼친다. 차림표선택은 Edit|Find in Text이고 지름건은 Ctrl+F이다.

• Print는 Print대화칸을 펼친다. 차림표선택은 File|Print이고 지름건은 Ctrl+P이다.

• Zoom in는 서체크기를 증가시킨다. 차림표선택은 View|Zoom in이고 지름건은 Ctrl++이다.

• Zoom out는 서체크기를 감소시킨다. 차림표선택은 View|Zoom out이고 지름건은 Ctrl+-이다.

• What's This?는 Qt Assistant특성의 서술을 제공한다. 차림표선택은 Help|What's This?이고 지름건은 Shift+F1이다.

나머지 도구띠단추들은 서표이며 구성환경에 따라 달라진다.

4. 차림표

1) File차림표

• File|Print는 Print대화칸을 펼친다.

• File|Exit는 Qt Assistant를 완료한다.

2) Edit차림표

• Edit|Copy는 선택된 본문을 오려둠판에 복사한다.

• Edit|Find in Text는 Find Text대화칸을 호출한다.

• Edit|Settings은 Settings대화칸을 펼친다.

3) View차림표

• View|Zoom in은 서체크기를 증가시킨다.

• View|Zoom out는 서체크기를 감소시킨다.

• View|Views|Sidebar는 가름띠의 표시를 절환한다.

• View|Views|Toolbar는 도구띠의 표시를 절환한다.

• View|Views|Line up는 도구띠의 도구띠단추들을 정렬한다.

4) Go차림표

• Go|Previous는 이전 페지를 현시한다.

• Go|Next는 다음 페지를 현시한다.

• Go|home은 홈페지로 간다.

또한 이 차림표는 추가적인 항목들을 가지고있다. 이것들은 환경구성에 따라 달라지는 미리 정의된 서표들이다.

5) Bookmarks차림표

Bookmarks|Add는 서표목록에 현재페지를 추가한다.

이 차림표는 추가항목들 즉 이미 만든 서표들을 가지고있다. 서표를 삭제하려면 가름띠의 Bookmarks타브로 이행한다.

5. 대화칸

1) Print대화칸

이 대화칸은 가동환경에 고유하다. 이것은 각종 인쇄기선택에 대한 호출을 주며 현재 페지를 인쇄하는데 쓰일수 있다.

2) Find Text대화칸

이 대화칸은 현재 페지에서 본문을 찾는데 리용된다. Find행편집칸에 찾으려는 본문을 입력한다. Whole words only검사칸을 설정하면 탐색은 오직 완전히 일치하는 단어들을 검색한다. 즉 이 검사칸을 설정하고 spin을 검색한다면 spinbox는 찾지 않지만 spin은 찾는다. Case sensitive검사칸을 설정하면 실례로 spin은 Spin과는 대조되지 않는다. Direction라지오선택중 하나를 선택하여 페지안의 현재위치로부터 정방향으로(Forward) 또는 역방향으로(Backward) 검색할수 있다. Find단추를 찰칵하여 검색하고(혹은 다시 검색한다) Close단추를 찰칵하여 완료한다.

3) Settings대화칸

Settings대화칸은 Qt Assistant를 자기의 요구에 맞게 설정하는데 리용된다. 대화칸에는 4개의 타브 즉 General Settings, Web Settings, PDF Settings 그리고 Profiles가 있다. Qt Assistant는 창문크기와 위치, 열려진 페지들을 비롯하여 쎄숀들사이의 환경설정을 기억한다. 매개 타브는 다음에 설명한다.

• General설정

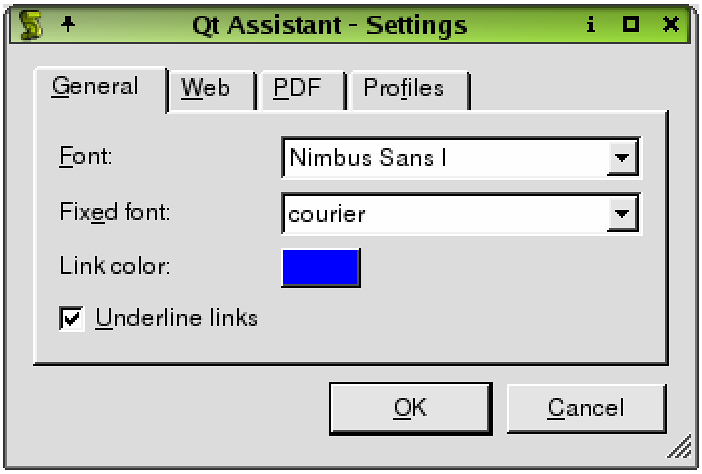


그림 3-5. General설정

Qt Assistant에서 사용한 기준서체를 변경하려면 Font복합칸에서 서체형을 선택한다. 새로운 고정폭의 서체를 선택하려면 실례로 코드부분을 표시하려면 Fixed font복합칸에서 서체형을 선택한다. 초본문련결색을 변경하려면 Link color색단추를 찰칵한다. 밑줄있는 련결을 바라지 않는다면 Underline links검사칸의 설정을 해제한다.

- Web설정

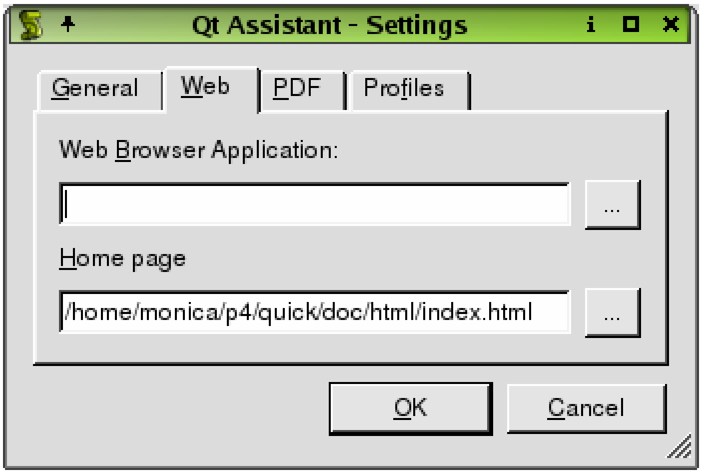


그림 3-6. Web설정

일부 페지들에는 외부웨브페지들에 대한 련결이 들어있다. 이 련결들을 현시하려면 웨브열람기를 지정해야 한다. Web Browser Application행편집칸에서 열람기의 실행가능파일의 이름을 입력한다. 또는 생략단추를 찰칵하여 Set Web Browser대화칸을 펼치고 사용하려는 웨브열람기를 찾을 때까지 항행한다. Save를 선택하여 선택을 받아들인다.

Qt Assistant의 기정홈페지를 변경하려면 Home Page행편집칸에 파일이름을 입력한다. 또는 생략단추를 찰칵하여 Set Homepage대화칸을 연다. 사용하려는 홈페지파일을 찾을 때까지 항행하고 Save를 눌러서 선택을 받아들인다.

• PDF설정

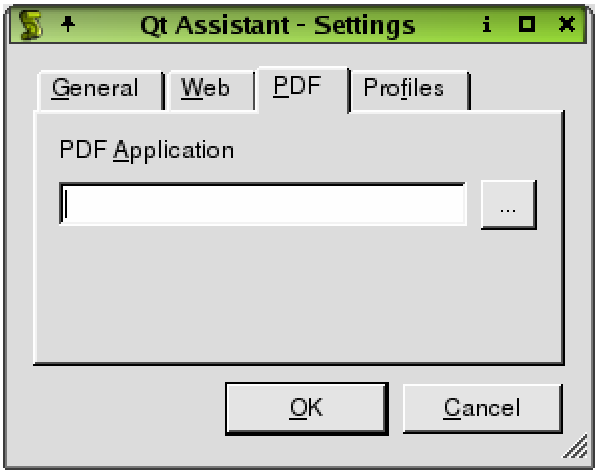


그림 3-7. PDF설정

일부 페지들은 PDF문서에 대한 련결을 포함한다. 이 련결을 현시하려면 PDF보기프로그람을 지정해야 한다. 행편집칸에서 PDF보기프로그람의 실행파일이름을 입력한다. 또는 생략단추를 찰칵하여 Set PDF Browser 대화칸을 열고 사용하려는 PDF보기프로그람을 찾을 때까지 항행한다. Save를 선택하여 선택을 받아들인다.

제4절. 완전본문검색

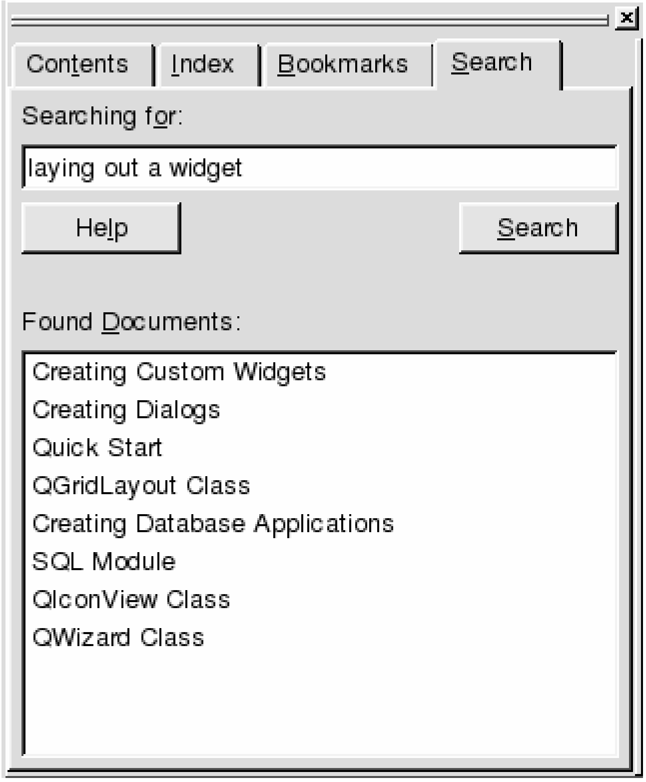


그림 3-8. 완전본문탐색

Qt Assistant는 강력한 완전본문탐색엔진을 제공한다. 어떤 단어나 본문을 검색하려면 Search타브를 선택한다. 그다음 검색하려는 본문을 입력하고 Enter를 누르거나 Search를 찰칵한다. 검색은 대소문자를 구별하지 않으므로 Foo, fOo 그리고 FOO 는 모두 같은것으로 취급된다. 다음은 일반탐색패턴의 실례들이다.

- deep – 단어deep를 포함하는 문서들을 모두 렬거한다.
- deep* -- deep로 시작하는 단어를 포함하는 문서들을 모두 렬거한다.
- deep copy -- deep와 copy를 둘다 포함하는 문서들을 모두 렬거한다.
- "deep copy" -- 구 deep copy를 포함하는 모든 문서들을 모두 렬거한다.

대리기호(*)는 인용표안에서 사용할수 없다.

발견한 문서목록은 문서들이 포함하는 탐색본문의 출현회수에 따라 순서로 놓이므로 출현회수가 제일 큰것이 우선 나타난다. 목록에서 임의의 문서를 선택하여 문서창문에 표시한다.

문서가 변경되였으면 즉 문서가 추가 또는 삭제되였으면 Qt Assistant는 재색인된다.

제5절. Qt Assistant의 전용화

Qt Assistant는 문서모임으로부터 문서를 추가하고 삭제하여 전용화할수 있다. 또한 Qt Assistant는 프로파일선택을 받아들이는데 이것은 그 속성을 변경할수 있게 한다. 실례로 기정시작페지와 응용프로그람그림기호를 들수 있다.

1. 기정문서모임의 수정

선택없이 시작할 때 Qt Assistant는 문서의 기정모임을 현시한다. Qt가 설치될 때 Qt Assistant안의 기정문서모임은 Qt Designer와 qmake와 같이 Qt와 함께 제공되는 도구들은 물론이고 Qt참고문서를 포함한다.

문서는 내용파일을 추가, 삭제함으로써 Qt Assistant로부터 추가 혹은 삭제할수 있다. 내용파일의 형식은 아래에 지정한다. 내용파일을 추가하려면 다음의 지령행선택 즉 -addContentFile docfile을 입력한다. 기정모임에서 내용파일을 삭제하려면 다음의 지령행선택 즉 -removeContentFile docfile을 입력한다. 실례로

```
1: > assistant -addContentFile file.dcf
2: > assistant
3: > assistant -removeContentFile file.dcf
```

1행에서는 file.dcf를 추가한다. 2행에서는 Qt Assistant를 기동한다. 이제는 기정모임이 doc파일 file.dcf로 확장된다. 3행에서는 기정문서모임에서 파일 file.dcf을 삭제함으로써 그후에 Qt Assistant를 사용할 때 이 파일을 포함하지 않게 한다.

- 문서내용파일형식

문서내용파일은 문서의 내용표와 색인용의 모든 중요열쇠단어들을 포함해야 한다. 또한 Qt Assistant도구띠에 현시되는 문서용그림기호를 계승할수 있다. 또한 문서에서 리용된 추가화상들의 여분의 등록부경로를 지정해야 한다.

모든 유효끄리표와 속성들을 사용하는 내용파일의 실례를 아래에 보여준다.

```
<assistantconfig  version="3.2.0">
  <DCF  ref="demo.html"  icon="handbook.png"  imagedir="../img/"
     title="Development  Demo  Handbook">
    <section  ref="./chap1/chap1.html"  title="Chapter1">
      <section  ref="./chap1/section1.html"  title="Section1">
        <keyword  ref="./chap1/section1.html#foo">foo</keyword>
        <keyword  ref="./chap1/section1.html#bla">bla</keyword>
        <section  ref="./chap1/section1.html#subsection1"  title="Subsection  1"/>
        <section  ref="./chap1/section1.html#subsection2"  title="Subsection  2"/>
```

```
      <section ref="./chap1/section1.html#subsection3" title="Subsection 3"/>
    </section>
    <section ref="./chap1/section2" title="Section2">
      <section ref="./chap1/section2.html#subsection1" title="Subsection 1"/>
      <section ref="./chap1/section2.html#subsection2" title="Subsection 2"/>
      <section ref="./chap1/section2.html#subsection3" title="Subsection 3"/>
    </section>
  </section>
  <section ref="./chap2/chap2.html" title="Chapter2">
    <keyword ref="./chap2/chap2.html#foo">foo</keyword>
    <section ref="./chap2/section1.html" title="Section1"/>
  </section>
</DCF>
</assistantconfig>
```

절은 필요한만큼 깊이 겹쌓을수 있다. 모든 참고는 련결되여야 한다.

주어진 절을 위한 keyword끄리표는 그 절안에 놓인 다른 절보다 앞에 있어야 한다.

refs속성안의 경로들은 항상 Unix형식(전방의 사선들)으로 씌여지며 문서내용파일 자체와 련관된다.

Qt 3.2.0판으로부터 파일형식에서 새로운 뿌리끄리표assistantconfig의 도입으로 한개 파일에서 여러개의 DCF끄리표를 지정할수 있게 되였다. Qt 3.2까지 사용한 낡은 문서내용파일형식은 여전히 유효하다.

2. 프로파일

프로파일은 Qt Assistant가 응용프로그람의 문서를 표시하기 위한 전문화된 방조도구로서 동작하게 한다. 프로파일을 리용하여 문서의 저자는 Qt Assistant의 제목, 응용프로그람그림기호, 판정보대화칸와 같은 속성들을 변경할수 있다. 또한 프로파일들은 Qt doc로부터 분리되는 전문화된 문서모임을 실행하는데 리용될수 있다. Qt Assistant는 다음의 속성들을 변경하여 전용화할수 있다.

- 이름 - 이 속성은 프로파일을 이름짓는데 사용된다. 다중프로파일을 Qt Assistant의 같은 설치에 사용한다면 이 파라메터는 프로파일에 고유한 환경설정을 따로 보관하는것이 중요하다. 속성이름은 name이다.
- 제목 - 이 속성은 Qt Assistant용의 제목을 지정하는데 쓰인다. 속성이름은 title이다.
- 응용프로그람그림기호 - 이 속성은 Qt Assistant응용프로그람그림기호로서 사용될 그림기호를 서술한다. 그림기호의 위치는 프로파일의 위치와 련관되여있다. 속성이름은 applicationicon이다.

• 시작페지 - 이 속성은 프로파일을 사용할 때 Qt Assistant가 처음에 어느 페지를 표시하는가를 지정한다. 보통 이것은 문서의 내용표를 포함하는 HTML파일이다. 또한 이 속성은 Qt Assistant의 기본사용자대면부에서 홈단추를 누를 때 가야 할 기정위치를 서술한다. 시작페지는 프로파일의 위치와 련관되여 지정된다. 속성이름은 startpage이다.

• 판정보차림표본문 - 이 속성은 Help차림표에 나타나는 본문 실례로 About Application을 서술한다. 속성이름은 aboutmenutext이다.

• URL정보 - 이 속성은 Help차림표에서 열려지는 About대화칸 실례로 About Application의 내용을 서술하는 HTML파일을 지적하는데 사용될수 있다. url은 프로파일의 위치와 련관되여 지정된다. 속성이름은 abouturl이다.

• Qt Assistant문서 - 이 속성은 Qt Assistant문서의 위치를 서술한다. 이것은 Qt Assistant가 완전본문검색방조와 Help차림표의 Qt Assistant Manual선택과 같은 자체의 방조를 제공하기때문에 요구된다. 위치는 프로파일의 위치와 련관한 등록부이다. 속성이름은 assistantdocs이다.

프로파일을 정의하기 위하여서는 보통 .adp로 생략된 Qt Assistant문서프로파일을 지정해야 한다. 프로파일은 우에서 서술한 문서내용파일의 확장이다. 형식으로 property 꼬리표들을 포함하는 profile꼬리표를 추가한다.

문서프로파일의 실례는 다음과 같다.

```
helpdemo.adp
<assistantconfig version="3.2.0">
  <profile>
    <property name="name">HelpExample</property>
    <property name="title">Help Example</property>
    <property name="applicationicon">logo.png</property>
    <property name="startpage">index.html</property>
    <property name="aboutmenutext">About Help</property>
    <property name="abouturl">../about.txt</property>
    <property name="assistantdocs">../../../doc/html</property>
  </profile>

  <DCF ref="index.html" icon="handbook.png" title="Help example">
    <section ref="./manual.html" title="How to use this Example">
      <keyword ref="./manual.html#installdocs">Install Docs</keyword>
      <keyword ref="./manual.html#onlydoc">Example Profile</keyword>
      <keyword ref="./manual.html#hide">Hide Sidebar</keyword>
```

```
        <keyword  ref="./manual.html#openqabutton">Open</keyword>
        <keyword  ref="./manual.html#closeqabutton">Close</keyword>
        <keyword  ref="./manual.html#display">Display</keyword>
     </section>
   </DCF>
</assistantconfig>
```

이 파일들은 XML파일이다. <, >, 그리고 &와 같은 문자들은 실체로서 씌여져야 한다. (실례로 <, >, &).

프로파일의 사용

프로파일을 리용하려면 선택 -profile filename으로 Qt Assistant를 실행한다. 이것은 파일에 지정된 프로파일을 적재하고 Qt Assistant를 전용화한다. 실례로 Qt Assistant를 우의 실례파일 helpdemo.adp를 리용하여 실행하려면 다음과 같이 지령을 실행한다.

```
> assistant -profile helpdemo.adp
```

Qt Assistant에 의하여 자기 응용프로그람들에서 프로파일을 사용하는 방법을 보여주는 HelpDemo실례를 Qt배포물로부터 참고하시오.

제4장. qmake

qmake는 각이한 콤파일러들과 가동환경용의 makefile을 작성하기 위하여 Trolltech가 개발한 도구이다.

수동적인 makefile작성은 복잡하여 오유를 범하기 쉬우며 특히 각이한 콤파일러들과 가동환경을 결합하여 여러개의 makefile들을 만들 때는 특히 더 복잡하다. 개발자들은 qmake를 리용하여 간단한 하나의 프로젝트파일을 만들고 qmake를 실행하여 적당한 makefile들을 생성한다. qmake는 콤파일러와 가동환경의 의존관계들을 모두 고려하여 개발자들이 마음놓고 자기 코드작성에만 집중할수 있게 한다. Trolltech는 Qt서고와 그와 함께 제공된 도구들에서 기본건설도구로서 qmake를 사용한다.

qmake는 또한 moc와 uic의 건설규칙들을 자동적으로 포함하여 Qt의 특수한 요구를 만족시킨다.

제1절. qmake설치

1. qmake설치

qmake는 Qt가 건설될 때 기정으로 건설된다.

이 절에서는 qmake를 수동적으로 건설하는 방법을 설명한다. 이미 qmake를 가지고있으면 2절로 넘어갈수 있다.

- qmake의 수동설치

Qt를 수동적으로 설치하기전에 다음과 같은 환경변수들을 설정해야 한다.

• QMAKESPEC

QMAKESPEC는 체계에서 사용하고있는 가동환경과 콤파일러결합에 따라서 설정되여야 한다. 실례로 Windows와 Microsoft Visual Studio를 사용하고있다면 이 환경변수를 win32-msvc로 설정하여야 한다. Solaris와 g++를 사용한다면 이 환경변수를 solaris-g++로 설정하여야 한다.

다음은 QMAKESPEC를 설정할 때 선택할수 있는 환경변수들을 보여준다.

aix-64 hpux-cc irix-032 netbsd-g++ solaris-cc unixware7-g++ aix-g++ hpux-g++ linux-cxx openbsd-g++ solaris-g++ win32-borland aix-xlc hpux-n64 linux-g++ openunix-cc sunos-g++ win32-g++ bsdi-g++ hpux-o64 linux-icc qnx-g++ tru64-cxx win32-msvc dgux-g++ hurd-g++ linux-kcc reliant-64 tru64-g++ win32-watc freebsd-g++ irix-64 macx-pbuilder reliant-cds ultrix-g++ win32-visa hpux-acc irix-g++ macx-g++ sco-g++ unixware-g hpux-acc irix-n32 solaris-64 unixware7-cc

환경변수는 qws/envvar로 설정되여야 하며 여기서 envvar는 다음것들중의 하나이다.

linux-arm-g++ linux-generic-g++ linux-mips-g++ linux-x86-g++ linux-freebsd-g++ linux-ipaq-g++ linux-solaris-g++ qnx-rtp-g++

·QTDIR

QTDIR는 Qt가 설치될 곳으로 설정되여야 한다. 실례로 c:\qt와 \local\qt

환경변수들이 일단 설정되면 qmake등록부 $QTDIR/qmake, 실례로 C:\qt\qmake로 들어간다. 그리고 자기의 콤파일러에 따라서 make 혹은 nmake를 실행한다.

make가 끝나면 qmake를 사용할 준비가 끝난다.

제2절. qmake사용법에 대한 소개

1. 프로젝트파일의 작성

qmake는 프로젝트파일(.pro)들에 보관된 정보를 리용하여 그것이 생성하는 makefile들에 포함해야 할것을 결정한다.

기본프로젝트파일은 응용프로그람에 대한 정보 례를 들면 응용프로그람을 콤파일하는데 요구되는 파일들과 사용하려는 환경설정을 포함한다.

여기에 프로젝트파일의 간단한 실례가 있다.

```
SOURCES = hello.cpp
HEADERS = hello.h
CONFIG += qt warn_on release
```

한행씩 간단히 설명한다.

```
SOURCES = hello.cpp
```

이 행은 응용프로그람을 실현하는 원천파일들을 지정한다. 이 경우에는 한개 파일 hello.cpp만 있다. 대다수 응용프로그람들은 여러 파일들을 요구하므로 이러한 경우에는 다음과 같이 한 행우에 공백으로 구분하여 모두 목록하는 방법으로 처리한다.

```
SOURCES = hello.cpp main.cpp
```

그렇지 않으면 매개의 파일을 다음과 같이 행을 바꾸어 개별적인 행들에 배치할수 있다.

```
SOURCES = hello.cpp \
          main.cpp
```

좀 장황한 방법은 다음과 같이 매개 파일을 따로따로 배치하는것이다.

```
SOURCES += hello.cpp
SOURCES += main.cpp
```

이 수법에서는 =보다도 +=를 사용하여 현존목록을 교체하지 않고 항상 새 파일을

안전하게 추가한다.

HEADERS행은 응용프로그람에서 사용하기 위하여 생성한 머리부파일을 지정하는데 쓰인다. 실례로

```
HEADERS += hello.h
```

원천파일들을 목록하는 임의의 수법을 머리부파일들에 사용할수 있다.

CONFIG행은 응용프로그람의 환경구성에 대한 qmake정보를 주는데 사용된다.

```
CONFIG += qt warn_on release
```

여기서는 이미 존재하는 환경선택들에 새로운 선택을 추가해야 하므로 +=를 사용한다. 이것은 모든 선택을 새로 주어진 선택들과 완전히 교체하는 =를 사용하는 경우보다 안전하다.

CONFIG행의 qt부분은 qmake에게 Qt를 사용하여 응용프로그람을 건설하고있다는것을 알린다. 이것은 qmake가 련결시에 Qt서고들과 련결하며 콤파일에 요구되는 머리부경로들을 포함시킨다는것을 의미한다.

CONFIG행의 warn_on부분은 qmake에게 경고가 출력되도록 콤파일러기발을 설정해야 한다는것을 알린다.

CONFIG행의 release부분은 qmake에게 응용프로그람을 출하판으로 건설해야 한다는것을 알린다. 개발시에 프로그람작성자들은 release를 debug와 교체하기를 더 좋아한다. 이에 대해서는 후에 론의한다.

프로젝트파일들은 일반본문 (즉 notepad, vim 혹은 xemacs와 같은 편집기를 사용)이고 .pro확장자로 보관되여야 한다. 실행가능프로그람의 이름은 프로젝트파일이름과 같아지지만 가동환경에 따라 적당한 확장자가 주어진다. 실례로 hello.pro라는 프로젝트파일은 Windows에서는 hello.exe, Unix에서는 hello를 생성한다.

2. makefile의 생성

프로젝트파일을 작성한 다음에는 makefile을 간단히 생성할수 있으며 작성자는 프로젝트파일을 작성한 위치로 가서 다음과 같이 입력한다.

즉 makefile들은 다음과 같이 .pro파일들로부터 생성된다.

```
qmake -o Makefile hello.pro
```

Visual Studio사용자인 경우에 qmake는 또한 .dsp파일들을 생성할수 있다. 실례로

```
qmake -t vcapp -o hello.dsp hello.pro
```

제3절. qmake련습

이 련습에서는 qmake사용법을 설명한다.

1. 간단히 시작하기

자기 응용프로그람의 기본실현을 방금 끝내고 다음과 같은 파일들을 창조하였다고 가정하자.

·hello.cpp

·hello.h

·main.cpp

qt/qmake/examples/tutorial에서 이 파일들을 찾을수 있다. 응용프로그람의 설치에 대하여 한가지 알아야 할것은 그것이 Qt로 씌여지는것이다. 우선 자기가 좋아하는 일반본문편집기를 리용하여 qt/qmake/tutorial에서 hello.pro라는 파일을 창조한다. 처음에 해야 할 일은 qmake에게 자기의 개발프로젝트의 부분인 원천파일과 머리부파일들에 대하여 알리는 행들을 추가하는것이다.

우선 프로젝트파일에 원천파일들을 추가한다. 그러자면 SOURCES변수를 사용해야 한다. SOURCES +=로 새 행을 시작하고 그 뒤에 hello.cpp를 넣는다. 즉

```
SOURCES  += hello.cpp
```

프로젝트안의 매개 원천파일에 대하여 다음행들로 끝날 때까지 이것을 반복한다.

```
SOURCES  += hello.cpp
SOURCES  += main.cpp
```

make형의 문법을 사용하기 좋아한다면 다음과 같이 새행확장기호를 리용하여 모든 파일을 한번에 렬거할수 있다.

```
SOURCES  = hello.cpp \
           main.cpp
```

이제는 프로젝트안에 원천파일들이 렬거되여있으므로 머리부파일들을 추가해야 한다. 원천파일들과 같은 방법으로 머리부파일들을 정확히 추가한다. 하지만 변수이름은 HEADERS이다.

이것을 수행한 다음 자기 프로젝트파일은 다음과 같아야 한다.

```
HEADERS  += hello.h
SOURCES  += hello.cpp
SOURCES  += main.cpp
```

목표이름은 자동적으로 설정되고 프로젝트파일과 같지만 가동환경에 적합한 뒤붙이가 있다. 실례로 프로젝트파일을 'hello.pro'라고 부른다면 목표는 Windows에서 'hello.exe', Unix에서 'hello'로 된다. 다른 이름을 사용하려면 그것을 프로젝트파일

에서 설정할수 있다

```
TARGET = helloworld
```

마지막 단계는 CONFIG변수의 설정이다. 이것이 Qt응용프로그람이므로 CONFIG행에 'qt'를 넣음으로써 qmake가 련결해야 할 관련서고들을 추가하게 하여 makefile에 moc와 uic용의 건설행들이 포함된다는것을 담보한다.

완료된 프로젝트파일은 다음과 같다.

```
CONFIG += qt
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
```

이제는 qmake를 사용하여 자기 응용프로그람을 위한 makefile을 생성할수 있다. 지령행에서 자기 응용프로그람등록부안에서 다음과 같이 입력한다.

```
qmake -o Makefile hello.pro
```

그다음 사용하는 콤파일러에 따라 make 혹은 nmake라고 입력한다.

2. 응용프로그람을 오유수정가능하게 하기

응용프로그람의 출하판은 오유수정기호나 다른 오유수정정보를 포함하지 않는다. 개발시에 관련정보를 가지는 응용프로그람의 오유수정판을 생성하는것이 좋다. 이것은 프로젝트파일안의 CONFIG변수에 'debug'를 추가하여 간단히 달성된다.

실례로

```
CONFIG += qt debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
```

makefile을 생성하기전에 qmake를 사용하여 자기 응용프로그람을 오유수정가능하게 할수 있다.

3. 가동환경에 고유한 원천파일들의 추가

몇시간 코드작성한 후에 자기 응용프로그람의 가동환경에 고유한 코드부분을 만들어 가동환경에 의존하는 코드를 분리하여 만들려고 결심할수 있다. 그러면 자기의 프로젝트파일에 포함해야 할 파일이 두개로 된다. 즉 hellowin.cpp와 hellounix.cpp를 들수 있다. SOURCES변수에 이것들을 추가하면 makefile에 두 파일이0 모두 넣어지므로 그렇게 할수 없다. 그러므로 여기서 할 일은 qmake를 실행하는 가동환경에 따라서 처리되는 유효범위를 리용하는것이다.

Windows용의 가동환경에 의존하는 파일에 추가해야 할 간단한 유효범위는 다음과

같다.

```
win32 {
   SOURCES += hellowin.cpp
}
```

그러므로 qmake를 Windows에서 실행한다면 원천파일목록에 hellowin.cpp가 추가된다. qmake를 다른 가동환경에서 실행한다면 단지 그것을 무시한다. 이제 남은 일은 unix의존파일용의 유효범위를 창조하는것이다.

그것을 수행하였을 때 자기의 프로젝트파일은 다음과 같아야 한다.

```
CONFIG += qt debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
   SOURCES += hellowin.cpp
}
unix {
   SOURCES += hellounix.cpp
}
```

makefile을 생성하기전에 qmake를 사용하여야 한다.

4. 파일이 존재하지 않으면 qmake를 중지하기

어떤 파일이 존재하지 않으면 makefile을 창조하지 않으려고 할수 있다. exists()함수에 의하여 파일이 있는가 검사할수 있다. error()함수를 사용함으로써 qmake의 처리를 중지시킬수 있다. 이것은 유효범위와 같은 수법으로 작업한다. 단순히 유효범위조건을 함수로 바꾸면 된다. main.cpp파일의 검사는 다음과 같다.

```
!exists(main.cpp) {
   error( "No main.cpp file found" )
}
```

!는 본문을 부정하는데 쓰인다. 즉 exists(main.cpp)는 파일이 존재하면 참이고 !exists(main.cpp)은 파일이 존재하지 않으면 참이다.

```
CONFIG += qt debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
```

```
    SOURCES += hellowin.cpp
}
unix {
    SOURCES += hellounix.cpp
}
!exists( main.cpp ) {
    error( "No main.cpp file found" )
}
```

makefile을 생성하기전에 qmake를 사용하여야 한다. 일시적으로 main.cpp의 이름을 변경한다면 통보문을 보게 되며 qmake는 처리를 중지한다.

5. 하나이상의 조건검사

Windows를 사용하며 지령행에서 자기의 응용프로그람을 실행할 때 qDebug()명령문을 볼수있게 하려고 한다고 가정한다. console설정하여 자기의 응용프로그람을 건설하지 않으면 출력을 볼수 없다. CONFIG행에 간단히 console을 넣음으로써 Windows에서 makefile이 이러한 설정을 가지게 할수 있다. 그러나 Windows에서 실행하고있는 조건에서 debug가 이미 CONFIG행에 있을 때 CONFIG행만 추가하려고 한다고 하자. 이것은 두개의 겹쌓인 유효범위를 사용할것을 요구한다. 즉 하나의 유효범위를 창조하고 그안에 다른것을 창조한다. 마지막 유효범위안에 처리하려는 설정내용을 다음과 같이 넣는다.

```
win32 {
    debug {
        CONFIG += console
    }
}
```

겹쌓인 유효범위들은 두점을 사용하여 모두 결합할수 있으므로 최종의 프로젝트파일은 다음과 같아진다.

```
CONFIG += qt debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
    SOURCES += hellowin.cpp
}
unix {
```

```
  SOURCES += hellounix.cpp
}
!exists( main.cpp ) {
  error( "No main.cpp file found" )
}
win32:debug {
  CONFIG += console
}
```

이로써 qmake에 대한 련습을 끝마친다. 이제는 자기의 개발프로젝트의 프로젝트파일을 쓸 준비가 되였다.

제4절. qmake에서 사용하는 용어들

qmake는 각이한 가동환경들에서 사용할 개발프로젝트용makefile들을 창조하기 위하여 Trolltech에서 제공하는 사용하기 편리한 도구이다. qmake는 불과 몇행의 정보로 makefile을 창조할수 있게 함으로써 makefile들의 생성을 단순화한다. qmake는 Qt로 작성하거나 작성하지 않는 임의의 쏘프트웨어프로젝트에 쓰일수 있지만 Qt개발을 지원하기 위한 추가적인 기능을 포함하고있다.

qmake는 프로젝트파일안의 정보에 기초하여 makefile을 창조한다. 프로젝트파일들은 개발자에 의해 창조된다. 프로젝트파일들은 보통 단순하지만 필요하다면 아주 복잡해질수 있다. qmake는 또한 프로젝트파일을 변경하지 않고도 Microsoft Visual studio용프로젝트들을 생성할수 있다.

1. qmake의 기본용어

1) QMAKESPEC환경변수

qmake를 makefile건설에 사용하기전에 QMAKESPEC환경변수를 체계에 사용되고있는 가동환경-콤파일러결합으로 설정해야 한다. QMAKESPEC환경변수는 가동환경과 콤파일러에 고유한 정보를 어디서 찾아야 하는가를 qmake에게 말한다. 이것은 정확한 서고를 사용한다는것과 생성된 makefile이 정확한 문법을 사용한다는것을 담보한다. 현재 유지된 가동환경-콤파일러결합의 목록은 qt/mkspecs에서 찾아볼수 있다. 자기의 환경변수를 렬거한 등록부들중 하나로 설정한다.

실례로 Windows에서 Microsoft Visual Studio를 사용하고 있다면 QMAKESPEC환경변수를 win32-msvc로 설정한다. Solaris에서 gcc를 사용하고있다면 자기의 QMAKESPEC환경변수를 solaris-g++로 설정한다.

qt/mkspecs안의 매개 등록부에는 가동환경과 콤파일러에 고유한 정보를 포함하는 qmake.conf파일이 있다. 이 설정들은 qmake에 의해 건설되는 프로젝트에 적용되며 자기가 전문가가 아닌 이상 수정하지 말아야 한다. 실레로 모든 응용프로그람들이 특정 서고에 련결해야 한다면 이 정보를 련관된 qmake.conf파일에 추가할수 있다.

2) 프로젝트(.pro)파일들

프로젝트파일은 응용프로그람의 makefile을 창조하는것과 련관된 자세한 정보를 qmake에게 알리는데 사용된다. 실례로 프로젝트파일에 넣어야 할 원천파일과 머리부파일들의 목록, 련결되여야 할 특수한 서고와 같은 응용프로그람에 고유한 환경구성, 혹은 특수한 머리부경로이다.

- #설명문

프로젝트파일에 설명문을 추가할수 있다. 설명문은 #기호로 시작하며 행끝까지 실행한다.

3) 형판

형판변수는 qmake에게 응용프로그람에 어떤 종류의 makefile이 생성되여야 하는가를 알린다. 다음의 선택이 가능하다.

- app - 응용프로그람을 건설하는 makefile을 창조한다. 이것은 기정이므로 형판이 지정되지 않으면 이것이 사용된다.
- lib - 서고를 건설하는 makefile을 창조한다.
- vcapp - 응용프로그람을 건설하는 Visual Studio프로젝트파일을 창조한다.
- vclib - 서고를 건설하는 Visual Studio프로젝트파일을 창조한다.
- subdirs - 이것은 지정된 등록부에 들어가는 makefile을 창조하는 특수한 형판로서 그 프로젝트파일의 makefile을 창조하고 그것에 대하여 make를 호출한다.

- app형판

app형판는 qmake에 응용프로그람을 건설하는 makefile을 생성할것을 알린다. 이 형판을 리용할 때 다음의 qmake체계변수들이 인식된다. 자기의 .pro파일에서 이것들을 사용하여 자기의 응용프로그람에 대한 정보를 지정해야 한다.

- HEADERS - 응용프로그람의 모든 머리부파일들의 목록.
- SOURCES - 응용프로그람의 모든 원천파일들의 목록.
- FORMS - 응용프로그람의 모든 .ui파일들(Qt Designer에 의해 창조된것)의 목록.
- LEXSOURCES - 응용프로그람의 모든 lex원천파일들의 목록.
- YACCSOURCES - 응용프로그람의 모든 yacc원천파일들의 목록.
- TARGET - 응용프로그람의 실행파일이름. 기정으로 이것은 프로젝트파일이름이다. (확장자는 자동적으로 추가된다.)
- DESTDIR - 실행가능한 목표가 배치되는 등록부.

• DEFINES – 응용프로그람에 필요한 앞처리정의들의 목록.
• INCLUDEPATH – 응용프로그람에 필요한 추가적인 머리부경로들의 목록.
• DEPENDPATH – 응용프로그람의 의존관계탐색경로.
• VPATH – 공급된 파일들을 찾기 위한 탐색경로.
• DEF_FILE – Windows만: 응용프로그람에 련결하려는 .def파일.
• RC_FILE – Windows만: 응용프로그람의 자원파일.
• RES_FILE – Windows만: 응용프로그람에 련결하려는 자원파일.

오직 값을 가지는 체계변수들만 사용할 필요가 있다. 실례로 특수한 INCLUDEPATH들을 가지지 않는다면 아무것도 지정할 필요가 없으며 qmake는 기정으로 필요한것을 추가한다. 실례로 실례프로젝트파일은 다음과 같을수 있다.

```
TEMPLATE = app
DESTDIR  = c:\helloapp
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
DEFINES += QT_DLL
CONFIG  += qt warn_on release
```

하나의 값을 가지는 항목들 실례로 형판 또는 목적등록부에 대하여 =를 사용하지만 여러값을 가지는 항목들에 대해서는 +=를 사용하여 그 형의 현존항목들에 추가한다. =를 사용하면 항목의 값이 새 값으로 교체되고 실례로 DEFINES=QT_DLL라고 썼으면 다른 모든 정의가 삭제된다.

– lib형판

lib형판는 qmake에게 서고를 건설하는 makefile을 생성하도록 한다. 이 형판을 리용할 때 app형판에 대하여 우에서 언급한 체계변수와 함께 VERSION변수가 유지된다. 자기의 .pro파일에서 이것들을 사용하여 서고에 대한 정보를 지정해야 한다.

• VERSION – 목표서고의 판번호, 실례로 2.3.1.

– subdirs형판

subdirs형판는 qmake에게 지정된 보조등록부안에 들어갈 makefile을 생성하고 그 등록부안에 프로젝트파일의 makefile을 생성하고 그것에 대하여 make를 호출해게 한다.

이 형판용으로 인식되는 유일한 체계변수는 SUBDIRS변수이다. 이 변수는 처리하려는 모든 보조등록부들의 목록을 포함한다. qmake가 프로젝트파일을 찾을수 있도록 보조등록부안의 프로젝트파일이 보조등록부와 같은 이름을 가지게 하는것이 본질이다. 실례로 보조등록부가 myapp라면 그 등록부안의 프로젝트파일은 그 등록부안에서 myapp.pro로 불리워야 한다.

4) CONFIG변수

config변수는 콤파일러가 사용하여야 할 선택들과 련결해야 할 서고들을 지정한다. 임의의것을 config변수에 추가할수 있으나 아래에 준 선택들은 qmake에 의하여 내부적으로 인식된다.

다음의 선택들은 어떤 콤파일러선택을 사용하는가를 조종한다.

- release – 응용프로그람을 출하방식으로 건설한다. 이것은 debug가 지정되면 무시된다.
- debug – 응용프로그람을 오유수정방식으로 건설한다.
- warn_on – 콤파일러는 될수록 많은 경고를 출력해야 한다. 이것은 warn_off가 지정되면 무시된다.
- warn_off – 콤파일러는 될수록 적은 경고를 출력해야 한다.

다음의 선택은 건설하려는 서고나 응용프로그람의 형을 정의한다.

- qt – 응용프로그람은 Qt응용프로그람이고 Qt서고에 련결되여야 한다.
- thread – 응용프로그람은 다중스레드응용프로그람이다.
- x11 – 응용프로그람은 X11응용프로그람 혹은 서고이다.
- windows – app형판만: 응용프로그람은 Windows창문응용프로그람이다.
- console – app형판만: 응용프로그람은 Windows콘솔응용프로그람이다.
- dll – lib형판만: 서고는 공유서고(dll)이다.
- staticlib – lib형판만: 서고는 정적서고이다.
- plugin – lib형판만: 서고는 플라그인이며 이것은 dll선택을 가능하게 한다.

실례로 자기의 응용프로그람이 Qt서고를 사용하고 그것을 오유수정가능한 다중스레드 응용프로그람으로 건설하려고 한다면 자기의 프로젝트파일은 다음과 같은 행을 가진다.

```
CONFIG += qt thread debug
```

=가 아니라 +=를 사용해야 하며 qmake는 Qt를 건설하는데 사용한 설정을 어떤 형의 Qt서고가 건설되였는가 하는 차림표로서 사용할수 없다.

2. 연산자

지금까지는 프로젝트파일에서 사용되고있는 =연산자와 += 연산자를 보았다. 사용할수 있는 연산자들은 더 있으나 그 일부는 기대한것보다 더 변경할 때에는 조심히 사용해야 한다.

① =연산자

이 연산자는 단순히 변수에 값을 대입하며 다음과 같이 사용된다.

```
TARGET = myapp
```

이것은 TARGET변수를 myapp로 설정한다. 이것은 이전에 설정된 TARGET를 삭제한다.

② +=연산자

이 연산자는 변수의 값목록에 값을 추가한다. 이것은 다음과 같이 사용된다.

```
DEFINES += QT_DLL
```

이것은 makefile에 넣어지는 앞처리정의의 목록에 QT_DLL를 추가한다.

③ −=연산자

이 연산자는 변수안의 값목록에서 값을 삭제한다. 그것은 다음과 같이 사용된다.

```
DEFINES -= QT_DLL
```

이것은 makefile에 넣어지는 앞처리정의의 목록에서 QT_DLL을 삭제한다.

④ *=연산자

이 연산자는 변수안의 값목록에 어떤 값이 존재하지 않으면 그것을 추가한다. 그것은 다음과 같이 사용된다.

```
DEFINES *= QT_DLL
```

QT_DLL은 앞처리정의들의 목록에 그것이 없는 경우에만 추가된다.

⑤ ~=연산자

이 연산자는 regexp와 일치하는 값들을 지정된 값으로 교체한다. 이것은 다음과 같이 사용된다.

```
DEFINES ~= s/QT_[DT].+/QT
```

이것은 QT가 있는 QT_D 혹은 QT_T로 시작하는 목록안의 값들을 삭제한다 .

3. 유효범위

유효범위는 'if'명령문과 비슷하다. 즉 일정한 조건이 참이면 유효범위안에서 설정내용이 처리된다. 유효범위는 다음과 같이 쓸수 있다.

```
win32 {
  DEFINES += QT_DLL
}
```

우의 코드는 qmake가 Windows가동환경에서 사용된다면 makefile에 QT_DLL정의를 추가한다. qmake가 Windows가 아닌 다른 가동환경에서 사용된다면 그 정의는 무시된다. 또한 다음과 같이 qmake에 의하여 한행의 대입을 수행할수도 있다.

```
win32:DEFINES += QT_DLL
```

실례로 Windows를 제외한 모든 가동환경에서 무엇인가 처리하려고 한다고 가정하자. 다음과 같이 유효범위를 부정하여 이것을 달성할수 있다.

```
!win32 {
  DEFINES += QT_DLL
}
```

CONFIG행의 항목도 유효범위이다. 실례로 다음과 같이 쓴다면

```
CONFIG += warn_on
```

warn_on라는 유효범위를 가질것이다. 이것은 특정한 환경구성에 필요되는 전용환경설정을 모두 잃지 않고 하나의 프로젝트에 대한 환경구성을 간단히 변경하게 한다. CONFIG행에 자체의 값들을 넣을수 있으므로 이것은 자기의 makefile들에 아주 강력한 환경구성을 제공한다. 실례로

```
CONFIG += qt warn_on debug
debug {
  TARGET = myappdebug
}
release {
  TARGET = myapp
}
```

우의 코드에서 두개 리용범위가 CONFIG행에 무엇이 넣어지는가에 따라 창조된다. 실례에서 debug는 config행우에 있으므로 TARGET변수가 myappdebug로 설정된다. release가 config행우에 있으면 TARGET변수가 myapp로 설정된다.

또한 일부 설정을 처리하기전에 두가지를 검사하는것이 가능하다. 실례로 가동환경이 Windows이고 스레드환경이 설정되는가 검사하려고 한다면 다음과 같이 쓰군 한다.

```
win32 {
  thread {
    DEFINES += QT_THREAD_SUPPORT
  }
}
```

겹쌓인 많은 유효범위를 간단히 쓰려면 다음과 같이 두점을 리용하여 유효범위들을 겹쌓을수 있다.

```
win32:thread {
  DEFINES += QT_THREAD_SUPPORT
}
```

일단 시험을 하였다면 또한 else/elseif조작을 수행할수도 있다. 이와 함께 복잡한 시험을 간단히 쓸수 있다. 이것은 특별한 else유효범위를 가지고 수행될수 있으며 우와 같이 두점으로 분리하여 다른 유효범위들과 결합될수 있다. 실례로

```
win32:thread {
  DEFINES += QT_THREAD_SUPPORT
} else:debug {
  DEFINES += QT_NOTHREAD_DEBUG
```

```
} else {
   message("Unknown configuration")
}
```

4. 변수

지금까지 보아온 변수들은 DEFINES, SOURCES, HEADERS와 같은 체계변수들이다. 자체의 변수들을 창조하여 유효범위안에서 그것들을 사용할수 있다. 자체의 변수를 창조하는것은 간단하다. 변수의 이름을 지정하고 거기에 무엇인가 할당한다. 실례로

```
MY_VARIABLE = value
```

자체의 변수들에 무엇을 수행하여야 하는가에 대한 제한은 없다. qmake는 어떤 유효범위에 대하여 변수들을 고찰할 필요가 없으면 무시한다.

변수이름앞에 $$를 놓음으로써 다른 변수에 현재 변수의 값을 할당할수 있다. 실례로

```
MY_DEFINES = $$DEFINES
```

그러면 MY_DEFINES변수는 프로젝트파일안에서 현재 DEFINES변수에 있는것을 포함한다. 이것은 또한 다음과 등가하다.

```
MY_DEFINES = $${DEFINES}
```

둘째 표기는 공백으로 구분함이 없이 변수를 다른 값으로 전개할수 있게 한다. qmake는 변수가 임의의것(Makefile로 직접 교체되는 $(VALUE) 등)을 포함하게 하며 적당히 보통 환경변수로서 전개하게 한다. 그러나 환경변수를 곧 교체할것을 요구한다면 $$()표기를 사용할수 있다. 실례로

```
MY_DEFINES = $$(ENV_DEFINES)
```

이것은 .pro파일을 해석할 때 MY_DEFINES를 환경변수 ENV_DEFINES의 값으로 설정한다. 추가적으로 교체하고있는 변수에서 기본함수를 호출할수 있다. 이 함수들(다음에 렬거하는것처럼 시험함수들과 혼돈하지 말아야 한다.)을 아래에 렬거한다.

① join(variablename, glue, before, after)

이것은 variablename의 값을 glue와 결합한다. 이 값이 비지 않았으면 값을 before의 앞에, 그리고 after의 뒤에 둔다. variablename는 유일하게 필요되는 마당이고 다른것들의 기정값은 빈 문자렬이다. glue, before 혹은 after안의 공백들을 부호화하여야 한다면 그것들을 인용해야 한다.

② prompt(question)

이것은 question을 현시하고 stdin으로부터 돌림값으로서 읽어들인다.

③ member(variablename, position)

이것은 목록의 position위치에 있는 variablename에 값을 배치한다. variablename의 값이 길지 않으면 이것은 빈 문자렬을 돌려준다. variablename은 유일하게 요구되는 마당이다. 위치가 지정되지 않으면 목록안의 첫값(0)으로 기정설정된다.

④ find(variablename, substr)

이것은 variablename안에서 substr와 일치하는 값들을 모두 배렬한다. substr는 물론 정규식일수 있으며 따라서 대조될것이다.

```
MY_VAR = one two three four
MY_VAR2 = $$join(MY_VAR, " -L", -L) -Lfive
MY_VAR3 = $$member(MY_VAR, 2) $$find(MY_VAR, t.*)
```

MY_VAR2는 '-Lone -Ltwo -Lthree -Lfour -Lfive'을 포함하며 MYVAR3는 'three two three'을 포함한다.

⑤ system(program_and_args)

이것은 실행된 프로그람의 stdout/stderr를 돌려주며 보통 기대한대로 그 문법을 해석한다. 이것을 리용하여 가동환경에 대한 정보를 질문할수 있다. 실례로

```
UNAME = $$system(uname -s)
contains( UNAME, [lL]inux ):message( This looks like Linux ($$UNAME) to me )
```

5. 시험함수

qmake는 단순하지만 강력한 시험을 진행하는 기본함수들을 제공한다. 이 시험은 우에서 서술한것처럼 유효범위의 위치에서 사용될수 있으며 일부 경우에는 시험값을 무시하고 자체의 시험함수를 사용하는것이 좋다.

① contains(variablename, value)

value가 variablename라는 변수에 보관한 값목록에 있으면 유효범위안의 환경설정들이 처리된다. 실례로

```
contains( CONFIG, thread ) {
   DEFINES += QT_THREAD_SUPPORT
}
```

thread가 CONFIG변수용값목록에 있으면 QT_THREAD_SUPPORT가 DEFINES변수안의 값목록에 추가된다.

② count(variablename, number)

number가 variablename라는 변수에 보관된 값들의 수와 일치하면 유효범위안의 환경설정들이 처리된다. 실례로

```
count( DEFINES, 5 ) {
   CONFIG += debug
}
```

③ error(string)

이 함수는 주어진 문자렬을 출력하고 qmake를 완료시킨다. 실례로

```
error( "An error has occured" )
```

본문 "An error has occured"이 콘솔에 현시되고 qmake는 완료된다.

④ exists(filename)

지정된 파일이 존재하면 유효범위안의 환경설정들이 처리된다. 실례로

```
exists( /local/qt/qmake/main.cpp ) {
  SOURCES += main.cpp
}
```

/local/qt/qmake/main.cpp이 존재하면 main.cpp가 원천파일목록에 추가된다. "/"가 가동환경에 관계없이 등록부분리기호로서 사용될수 있다는것을 알아두시오.

⑤ equals(variable, value)

지정된 변수가 넘긴 값과 같다면 그 유효범위가 처리된다. 실례로

```
NUMBERS = 1 2 3
equals( NUMBERS, 3 4 5 ) {
   message("The numbers are equal")
}
```

"1 2 3"이 "3 4 5"와 같지 않으므로 통보문이 현시되지 않는다. 모든 함수들에서와 같이 값인수로서 전개된 변수를 넘길수 있다. (즉 $$NUMBERS).

⑥ include(filename)

filename의 내용은 이 시점에서 프로젝트파일에 포함되므로 지정된 파일안의 환경설정들이 처리된다. 이 실례는 다음과 같다.

```
include( myotherapp.pro )
```

myotherapp.pro프로젝트파일안의 설정들이 지금 처리된다.

⑦ isEmpty(variablename)

이것은 count(variablename, 0)의 사용과 등가하다. variablename라는 변수가 요소를 가지지 않으면 유효범위안의 환경설정들이 처리된다. 실례로

```
isEmpty( CONFIG ) {
  CONFIG += qt warn_on debug
}
```

⑧ message(string)

이 함수는 단순히 콘솔에 통보문을 출력한다.

```
message( "This is a message" )
```

본문 "This is a message"이 콘솔에 출력되고 프로젝트파일의 처리가 수행된다.

⑨ system(command)

지정된 지령이 수행되여 완료코드 1을 돌려주면 유효범위안의 환경설정들이 처리된다. 실례로

```
system( ls /bin ) {
  SOURCES += bin/main.cpp
  HEADERS += bin/main.h
}
```

그러므로 지령 ls /bin이 1을 돌려주면 bin/main.cpp가 원천목록에 추가되고 bin/main.h가 머리부목록에 추가된다.

⑩ infile(filename, var, val)

이 함수는 파일 filename (qmake자체에 의해 해석될 때)이 값 val을 가지는 변수 var를 포함한다면 성공한다. 또한 제3인수(val)를 넘기지 않을수도 있으며 함수는 var가 파일에서 할당되였는가만 검사한다.

제5절. 사전번역된 머리부의 사용

사전콤파일된 머리부들은 안전한 코드본체를 콤파일하고 코드의 콤파일된 상태를 2진파일에 보관할 목적으로 일부 콤파일러들에 유지된 기능이다. 다음에 콤파일할 때 콤파일러는 보관된 상태를 적재하고 지정된 파일의 콤파일을 계속한다. 다음에 콤파일할 때 안전한 코드가 재콤파일을 요구하지 않으므로 더 빨라진다.

qmake는 일부 가동환경에서 사전번역된 머리부(PCH)와 다음과 같은 건설환경을 지원한다.

- Windows
  - o nmake
  - o Dsp 프로젝트 (VC 6.0)
  - o Vcproj 프로젝트 (VC 7.0 & 7.1)
- Mac OS X
  - o Makefile
  - o Xcode
  - o GCC 3.3이상
- Unix
  - o GCC 3.4이상

1. PCH를 자기 프로젝트에 추가

1) 사전콤파일된 머리부파일의 내용

사전콤파일된 머리부는 자기 프로젝트에서 완전히 안정하고 정적인 코드를 포함해야 한다. 전형적인 PCH는 다음과 같다.

stable.h

```
/* Add C includes here */
#if defined __cplusplus
/* Add C++ includes here */
#include <stdlib>
#include <iostream>
#include <vector>
#include <qapplication.h> // Qt includes
#include <qpushbutton.h>
#include <qlabel.h>
#include "thirdparty/include/libmain.h"
#include "my_stable_class.h"
...
#endif
```

C파일용사전콤파일된 머리부파일이 C++코드를 포함할수 없으므로 사전콤파일된 머리부파일은 CPP머리부로부터 C머리부를 분리할것을 요구한다.

2) 프로젝트선택

자기 프로젝트가 PCH를 사용하게 하기 위하여 자기의 프로젝트설정(.pro)에서 변경해야 할 유일한것은 PRECOMPILED_HEADER선택을 포함하는것이다.

```
PRECOMPILED_HEADER = stable.h
```

나머지는 qmake가 조종하여 사전콤파일된 머리부파일의 창조와 사용을 담보한다. HEADERS에 사전콤파일된 머리부파일을 포함할 필요는 없다. 환경구성이 PCH를 유지한다면 qmake가 이것을 수행한다.

사전콤파일된 머리부를 유지하는 모든 가동환경은 환경선택 precompile_header모임을 가진다. 이 선택을 리용하여 자기의 .pro파일에서 조건블로크들을 삽입하여 PCH를 리용할 때 설정값들을 추가할수 있다. 실례로

```
precompile_header:!isEmpty(PRECOMPILED_HEADER) {
   DEFINES += USING_PCH
}
```

2. 례제프로젝트

qt/qmake/examples/precompile등록부에서 다음과 같은 원천코드를 찾아볼수 있다.

① mydialog.ui

```
<!DOCTYPE UI><UI version="3.3" stdsetdef="1">
<class>MyDialog</class>
```

```
<widget class="QDialog">
   <property name="name">
      <cstring>MyDialog</cstring>
   </property>
   <property name="caption">
      <string>Mach 2!</string>
   </property>
   <vbox>
      <widget class="QLabel">
         <property name="name">
            <cstring>aLabel</cstring>
         </property>
         <property name="text">
            <string>Join the life in the fastlane; - PCH enable your project today! -</string>
         </property>
      </widget>
      <widget class="QPushButton">
         <property name="name">
            <cstring>aButton</cstring>
         </property>
         <property name="text">
            <string>&amp;Quit</string>
         </property>
         <property name="accel">
            <string>Alt+Q</string>
         </property>
      </widget>
   </vbox>
</widget>
</UI>
```

② stable.h

```
/* Add C includes here */
#if defined __cplusplus
/* Add C++ includes here */
```

```
  # include <iostream>
  # include <qapplication.h>
  # include <qpushbutton.h>
  # include <qlabel.h>
  #endif
```

③ myobject.h

```
#include <qobject.h>
class MyObject : public QObject
{
public:
    MyObject();
    ~MyObject();
};
```

④ myobject.cpp

```
#include <iostream>
#include <qobject.h>
#include "myobject.h"

MyObject::MyObject() : QObject()
{
    std::cout << "MyObject::MyObject()\n";
}
```

util.cpp

```
void util_function_does_nothing()
{
    // Nothing here...
    int x = 0;
    ++x;
}
```

⑤ main.cpp

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qlabel.h>
#include "myobject.h"
```

```
#include "mydialog.h"
int main(int argc, char **argv)
{
   QApplication app(argc, argv);
   MyObject obj;
   MyDialog dia;
   app.setMainWidget( &dia );
   dia.connect( dia.aButton, SIGNAL(clicked()), SLOT(close()) );
   dia.show();
   return app.exec();
}
```

⑥ precompile.pro

```
#############################################
# Example for using Precompiled Headers
#############################################
TEMPLATE   = app
LANGUAGE   = C++
CONFIG     += console precompile_header

# Use Precompiled headers (PCH)
PRECOMPILED_HEADER   = stable.h

HEADERS   += stable.h \
         myobject.h
SOURCES   += main.cpp \
         myobject.cpp \
         util.cpp
FORMS     = mydialog.ui
```

제6절. qmake지령들

이것은 교차가동환경makefile생성편의프로그람 qmake에서 사용하는 모든 지령행 선택과 환경구성, 내부변수들의 자세한 색인이다.

다음에 서술하는 변수, 함수들과 함께 qmake프로젝트파일에는 주석문을 포함할수도 있다. 주석은 #기호로 시작하여 행끝까지이다.

1. 지령행선택

1) 문법

qmake [options] files

2) 선택

다음의 선택들은 qmake의 지령행에 지정될수 있다.

-o file

qmake출력은 file에로 진행된다. 이 인수를 지정하지 않으면 qmake는 적당한 이름으로 추측한다. '-'가 지정되면 출력은 stdout에로 진행된다.

-unix

qmake는 unix방식에서 실행한다. 이 방식에서 Unix파일명명 및 경로관례를 사용하며 추가적으로 unix(유효범위로서)용의 시험이 계속된다. 이것은 모든 Unix에서 기정방식이다.

-macx

qmake는 Mac OS X방식에서 실행한다. 이 방식에서 Unix파일명명 및 경로관례를 사용하며 추가적으로 macx(유효범위로서) 용의 시험이 계속된다. 이것은 Mac OS X에서 기정방식이다..

-win32

qmake는 win32방식에서 실행한다. 이 방식에서 Windows파일명명 및 경로관례를 사용하며 추가적으로 win32(유효범위로서)용의 시험이 계속된다. 이것은 Windows에서 기정방식이다..

-d

qmake는 쓸모있는 오유수정정보를 출력한다.

-t tmpl

qmake는 .pro파일이 처리된 후에 tmpl을 가지는 TEMPLATE변수설정을 무시한다.

-tp prefix

qmake는 TEMPLATE변수에 앞붙이를 추가한다.

-help

qmake는 이 특성들을 조사하고 쓸모있는 방조를 준다.

또한 자기의 프로젝트파일에서 문제를 찾는데 도움을 줄수 있는 경고선택들이 있다.

-Wall

여기서 qmake는 알려진 모든 경고를 설정한다.

-Wnone

qmake는 어떤 경고정보도 생성하지 않는다.

-Wparser

qmake는 구문해석경고만 생성하며 이것은 자기의 .pro파일들에서 일반적인 함정들과 잠재하는 문제들을 경고한다.

-Wlogic

qmake는 다시 일반적인 함정들과 잠재하는 문제들에 대하여 경고한다. 이것은 파일이 파일목록에 배치되는가, 파일을 찾을수 없는가 등에 대한 검사를 포함할수 있다. 그러나 제한은 없다.

qmake는 두가지 서로 다른 방식의 조작을 유지한다. 첫째 방식은 기정으로서 makefile생성이다. 이 방식에서 qmake는 하나의 .pro파일을 가지며 그것을 makefile로 바꾼다. makefile의 창조는 이 참고차림표서에서 설명하며 거기에는 .pro파일들을 생성하는 다른 방식이 있다.

이 방식들을 절환하려면 첫 인수에 사용하려는 방식을 지정해야 한다. 방식을 지정하지 않으면 qmake는 makefile방식으로 가정한다. 쓸수 있는 방식들은 다음과 같다.

-makefile

qmake출력은 makefile(Makefile방식)이다.

-project

qmake출력은 프로젝트파일 (Project파일방식)이다.

- Makefile방식

Makefile방식에서 qmake는 makefile을 생성한다. 이 방식에서는 추가적으로 다음과 같은 인수들을 공급할수 있다.

-after

qmake는 지령행에서 지정된 파일들뒤에 주어진 인수들을 처리한다.

-nocache

qmake는 .qmake.cache파일을 무시한다.

-nodepend

qmake는 의존관계정보를 생성하지 않는다.

-cache file

qmake는 file을 캐쉬파일로서 사용하며 다른 .qmake.cache파일을 발견하면 그것

을 무시한다.

-spec spec

qmake는 spec를 가동환경콤파일러정보에로의 경로로 사용하며 QMAKESPEC가 무시된다.

files인수는 공백으로 구분된 하나이상의 프로젝트파일들의 목록일수 있다. 또한 여기서 지령행에 qmake대입을 넘길수 있으며 그것들은 모든 파일들을 지정하기전에 처리된다. 실례로

qmake -makefile -unix -o Makefile "CONFIG+=test" test.pro

그러나 자기의 변수들을 파일들이 지정된 후에 처리하려고 한다면 -after인수를 넘길수 있다. 이것을 지정하면 지령행에서 -after선택뒤의 모든 대입은 지정된 파일들이 해석된 후까지 연기된다.

이것은 Unix경로이름이 있는 test.pro로부터 Makefile을 생성한다. 그러나 이러한 인수들의 대부분은 그것들이 기정일 때 필요하지 않다. 그러므로 행은 Unix에서 다음과 같이 간단화된다.

qmake "CONFIG+=test" test.pro

Projectfile방식

Projectfile방식에서 qmake는 프로젝트파일을 생성한다. 또한 이 방식에서는 다음의 인수들을 제공할수 있다.

-r

qmake는 공급된 등록부들을 재귀적으로 조사한다.

-nopwd

qmake는 원천코드용현재작업등록부를 조사하지 않고 지정된 files만 사용한다.

files인수는 파일이나 등록부들의 목록일수 있다. 등록부가 지정되면 그것은 DEPENDPATH변수에 포함되며 그로부터 관련된 코드가 생성된 프로젝트파일에 포함된다. 파일이 주어지면 확장자에 따라서 정확한 변수에 들어간다. (즉 .ui파일들은 FORMS에 들어가고 .cpp파일들은 SOURCES에 들어간다.) 여기서 지령행에 대입을 넘길수도 있으며 그때 이 대입들은 생성된 .pro파일의 마감에 배치된다.

2. 체계변수

1) 흔히 사용하는 체계변수들

다음과 같은 변수들은 qmake가 인식하며 프로젝트파일을 창조할 때 제일 흔히 쓰인다.

CONFIG

CONFIG변수는 프로젝트환경구성과 콤파일러선택들을 지정한다. 값들은 qmake에 의해 내적으로 인식되며 특별한 의미를 가진다. 값들은 다음과 같다.

다음의 CONFIG값들은 콤파일기발들을 조종한다.

• release – 최적화를 허용하여 콤파일한다. debug가 지정되면 무시된다.

• debug – 오유수정선택을 허용하여 콤파일한다.

• warn_on – 콤파일러는 보통보다 경고를 더 많이 발생해야 한다. warn_off가 지정되면 무시된다.

• warn_off – 콤파일러는 엄중경고만 발생해야 한다.

다음의 선택들은 응용프로그람과 서고형을 정의한다.

• qt – 목표는 Qt응용프로그람 및 서고로서 Qt머리부파일 및 서고를 요구한다. Qt서고를 위한 적당한 머리부와 서고경로들이 프로젝트에 자동적으로 추가된다.

• opengl – 목표는 OpenGL(혹은 Mesa)머리부와 서고들을 요구한다. 이 서고들을 위한 적당한 머리부와 서고경로들은 프로젝트에 자동적으로 추가된다.

• thread – 목표는 다중스레드 응용프로그람이나 서고이다. 적당한 정의 및 콤파일러기발들이 프로젝트에 자동적으로 추가된다.

• x11 – 목표는 X11응용프로그람이나 서고이다. 적당한 머리부경로와 서고들이 프로젝트에 자동적으로 추가된다.

• windows – 목표는 Win32창문응용프로그람(app만)이다. 적당한 머리부경로들과 콤파일러기발, 서고들이 프로젝트에 자동적으로 추가된다.

• console – 목표는 Win32콘솔응용프로그람(app만)이다. 적당한 머리부경로와 콤파일러기발, 서고들이 프로젝트에 자동적으로 추가된다.

• dll – 목표는 공유목적파일 및 DLL이다. 적당한 머리부경로와 콤파일러기발, 서고들이 프로젝트에 자동적으로 추가된다.

• staticlib – 목표는 정적서고(lib만)이다. 적당한 콤파일러기발들이 프로젝트에 자동적으로 추가된다.

• plugin – 목표는 플라그인(lib만)이다. 이것은 물론 dll을 허용한다.

다음의 선택들을 콤파일러기발들을 설정하는데 사용한다.

• exceptions – 례외유지를 허용한다.

• rtti – RTTI유지를 허용한다.

• stl – STL유지를 허용한다.

다음의 선택들은 가동환경이나 형판에 따라서 특정한것들을 정의한다.

• flat – vcapp형판을 사용할 때 이것은 모든 원천파일들을 원천그룹에 넣으며 머리부파일들은 머리부그룹에 넣는다. 이때 그것들이 상주하는 등록부는 무시한다. 이 선택을 해제하면 파일들이 상주하는 등록부에 따라서 원천 및 머리부그룹안의 파일들을 묶어놓는다. 이것은 기정으로 선택되여있다.

CONFIG변수도 역시 범위를 해결할 때 검사된다. 이 변수에 아무것이나 대입할수 있다.

실례로

```
CONFIG += qt console newstuff
...
newstuff {
      SOURCES += new.cpp
      HEADERS += new.h
}
```

DEFINES

qmake는 이 변수의 값을 콤파일러의 C앞처리기마크로(-D선택)로서 추가한다.

실례로

```
DEFINES += USE_MY_STUFF QT_DLL
```

DEF_FILE

이것은 app형판을 사용할 때 Windows에서 사용된다.

.def파일을 지정하여 프로젝트에 포함되게 한다.

DESTDIR

목표파일을 넣으려는 곳을 지정한다.

실례로

```
  DESTDIR = ../../lib
```

DLLDESTDIR

목표dll을 복사하려는곳을 지정한다.

HEADERS

프로젝트용머리부파일들을 정의한다.

qmake는 지정된 머리부용의존관계정보를 생성한다. (그렇지 않으면 -nodepend가 지령행에 지정된다.) qmake는 또한 moc가 이 머리부들안의 클라스들에서 요구되는가를 자동적으로 탐지하고 적당한 의존관계와 파일들을 moc파일들을 생성하여 련결하려는 프로젝트에 추가한다.

실례로

```
HEADERS = myclass.h \
          login.h \
          mainwindow.h
```

또한 SOURCES를 참고.

INCLUDEPATH

이 변수는 프로젝트를 콤파일할 때 탐색되여야 할 #include등록부들을 지정한다. 등록부 분리기호로서 ;이나 공백을 사용한다.

실례로

INCLUDEPATH = c:\msdev\include d:\stl\include

FORMS

이 변수는 .ui파일들(Qt Designer참고)을 지정하여 콤파일하기전에 uic에 의해 처리되게 한다. .ui파일들을 건설하는데 요구되는 모든 의존관계들, 머리부와 원천파일들이 프로젝트에 자동적으로 추가된다.

실례로

```
FORMS = mydialog.ui \
      mywidget.ui \
      myconfig.ui
```

폼들을 += 연산자로 지정하지 말아야 한다. 그것은 이 문법이 Qt Designer에 의해 완전히 지원되지 않기때문이다.

LEXSOURCES

이 변수는 lex원천파일들의 목록을 포함한다. 모든 의존관계들, 머리부와 원천파일들이 lex파일을 건설할 때 프로젝트에 자동적으로 추가된다.

실례로

LEXSOURCES = lexer.l

LIBS

이 변수는 프로젝트에 련결할 서고목록을 포함한다. Unix관례 -L/-l 기발들에 친숙되여있으면 교차가동환경방식에서 그 기발들을 사용하는것은 자유이고 qmake는 Windows에서 이 서고들을 사용하여 정확한 일을 수행한다. (즉 이것은 서고의 완전경로를 련결프로그람에 넘기는것을 의미한다.) 이에 대한 유일한 제한은 qmake가 -l lib가 있는 등록부를 찾을수 있도록 서고가 존재해야 한다는것이다.

실례

unix:LIBS += -lmath -L/usr/local/lib

win32:LIBS += c:\mylibs\math.lib

MOC_DIR

이 변수는 모든 중간의 moc파일들이 배치되야 할 등록부를 지정한다.

실례로

unix:MOC_DIR = ../myproject/tmp

win32:MOC_DIR = c:\myproject\tmp

OBJECTS_DIR

이 변수는 모든 중간의 목적파일들이 배치되야 할 등록부를 지정한다.

실례로

unix:OBJECTS_DIR = ../myproject/tmp

win32:OBJECTS__DIR = c:\myproject\tmp

UI_DIR

이 변수는 uic가 만드는 모든 중간파일들이 배치되야 할 등록부를 지정한다. 이 변수는 UI_SOURCES_DIR와 UI_HEADERS_DIR를 둘다 무시한다.

실례로

unix:UI_DIR = ../myproject/ui

win32:UI_DIR = c:\myproject\ui

UI_HEADERS_DIR

이 변수는 uic가 만드는 모든 선언파일들이 배치되야 할 등록부를 지정한다.

실례로

unix:UI_HEADERS_DIR = ../myproject/ui/include

win32:UI_HEADERS_DIR = c:\myproject\ui\include

UI_SOURCES_DIR

이 변수는 uic가 만드는 모든 실현파일들이 배치되야 할 등록부를 지정한다.

실례로

unix:UI_SOURCES_DIR = ../myproject/ui/src

win32:UI_SOURCES_DIR = c:\myproject\ui\src

REQUIRES

이것은 qmake가 처리하는 특수변수이다. 이 변수의 내용은 이 변수가 할당될 때 CONFIG에 없으면 어떤 의존관계들(REQUIRES에 할당된 값들)을 빠뜨렸는가를 서술하는 최소의 makefile이 생성된다.

이것은 주로 Qt의 건설체계에서 실례들을 건설하는데 사용된다.

SOURCES

이 변수는 프로젝트안의 모든 원천파일들의 이름을 포함한다.

실례로

```
SOURCES = myclass.cpp \
        login.cpp \
        mainwindow.cpp
```

또한 HEADERS를 참고.

SUBDIRS

이 변수는 'subdirs' TEMPLATE에서 사용될 때 프로젝트파일에서 찾으려는 모든 보조등록부들의 이름을 포함한다.

실례로

SUBDIRS = kernel \
tools

TARGET

이것은 목표파일의 이름을 지정한다.

실례로

TEMPLATE = app

TARGET = myapp

SOURCES = main.cpp

우의 프로젝트파일은 unix에서 'myapp', windows 에서 'myapp.exe'라는 이름의 실행파일을 생성한다.

TEMPLATE

이 변수는 프로젝트를 생성할 때 사용하려는 형판이름을 포함한다. 허용값들은 다음과 같다.

- app - 응용프로그람들을 건설하기 위한 makefile을 창조한다(기정값).
- lib - 서고를 건설하기 위한 makefile을 창조한다.
- subdirs - 보조등록부들안의 목표들을 건설하기 위한 makefile을 창조한다.
- vcapp - win32전용. Visual Studio용응용프로그람 프로젝트파일을 창조한다.
- vclib - win32전용. Visual Studio용서고프로젝트파일을 창조한다.

실례

TEMPLATE = lib

SOURCES = main.cpp

TARGET = mylib

-t지령행선택에서 새로운 형판형을 지정하여 그 형판을 무시할수 있다. 이것은 .pro파일이 처리된 후에 형판형을 무시한다. 형판형을 사용하여 프로젝트건설방법을 결정하는 .pro파일들에서 지령행에서 -t 선택을 사용하기보다 TEMPLATE를 선언하는것이 필요하다.

VERSION

이 변수는 'lib' TEMPLATE 가 지정되면 서고의 판번호를 포함한다.

실례로

VERSION = 1.2.3

DISTFILES

이 변수는 dist목표에 포함해야 할 파일목록을 포함한다. 이 기능은 UnixMake지정에 의해서만 유지된다.

실례로

DISTFILES += ../program.txt

YACCSOURCES

이 변수는 프로젝트에 포함해야 할 yacc원천파일들의 목록을 포함한다. 모든 의존관계, 머리부와 원천파일들은 프로젝트에 자동적으로 포함된다.

실례로

YACCSOURCES = moc.y

2) 드문히 사용하는 체계변수들

다음과 같은 변수들도 역시 qmake가 인식하지만 내적으로 쓰이거나 아주 드문히 쓰인다.

DESTDIR_TARGET

이 변수는 qmake에 의해 내적으로 설정되며 주로 끝에 추가된 TARGET변수를 가지는 DESTDIR변수이다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

DSP_TEMPLATE

이 변수는 qmake에 의해 내적으로 설정되며 생성된 dsp파일들이 기초하기 위한 dsp형판이 보관되는곳을 지정한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

LEXIMPLS

이 변수는 lex파일들의 목록을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

LEXOBJECTS

이 변수는 중간의 lex목적파일들의 이름을 포함한다. 이 변수의 값은 전형적으로 qmake에 의하여 조종되며 드문히 수정할 필요가 있다.

LITERAL_HASH

이 변수는 변수선언에서 파일이름의 부분으로 혹은 외부응용프로그람에 넘긴 문자렬안에서 상수하쉬(hash)기호(#)가 요구될 때마다 사용된다.

실례

```
# To include a literal hash character, use the $$LITERAL_HASH variable:
urlPieces = http://doc.trolltech.com/3.3/qmake-manual-8.html LITERAL_HASH
message($$join(urlPieces, $$LITERAL_HASH))
```

이러한 방법으로 LITERAL_HASH를 사용함으로써 #기호를 콘솔에 출력하기 위한 message()함수용URL을 구성하는데 쓰일수 있다.

MAKEFILE

이 변수는 프로젝트건설에 대한 의존관계정보를 출력할 때 qmake 가 사용하는

makefile의 이름을 지정한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

MAKEFILE_GENERATOR

이 변수는 makefile을 생성할 때 사용할 makefile생성기의 이름을 포함한다. 이 변수의 값은 전형적으로 qmake에 의하여 조종되며 드문히 수정할 필요가 있다.

OBJECTS

이 변수는 SOURCES변수로부터 생성된다. 매개 원천파일의 확장자는 .o (Unix) 혹은 .obj (Win32)로 교체된다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

OBJMOC

이 변수는 Q_OBJECT마크로를 포함하는 파일들을 찾을수 있으면 qmake에 의해 설정된다. OBJMOC는 모든 중간moc목적파일들의 이름을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

PRECOMPILED_HEADER

이 변수는 사전콤파일된 머리부파일을 창조하기 위한 머리부파일을 지적하여 프로젝트의 콤파일속도를 증가시킨다. 사전콤파일된 머리부들은 현재 일부 가동환경에서만 유지되여있다. (Windows - 모든 MSVC프로젝트형, Mac OS X - X코드, Makefile, UNIX - gcc 3.3이상.)

다른 가동환경들에서 이 변수는 아래에 언급하는바와 같이 다른 의미를 가진다.

이 변수는 사전콤파일단계의 정렬(moc에서와 같이)을 요구하는 머리부파일들의 목록을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE

이 변수 qmake프로그람 그자체의 이름을 포함하며 생성된 makefile들에 배치된다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKESPEC

이 변수 makefile들을 생성할 때 사용할 qmake환경구성의 이름을 포함한다. 이 변수의 값은 전형적으로 qmake에 의하여 조종되며 드문히 수정할 필요가 있다. 그대신에 QMAKESPEC 환경변수를 사용한다.

QMAKE_APP_FLAG

이 변수는 'app' TEMPLATE 가 지정되지 않으면 비여있다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다. 그대신 다음의것을 사용한다.

```
app {
    #conditional code for 'app' template here
}
```

QMAKE_APP_OR_DLL

이 변수는 'app' 혹은 'dll' TEMPLATE 가 지정되지 않으면 비여있다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_AR_CMD

이것은 Unix가동환경에서만 사용된다

이 변수는 어카이브들을 창조하고 수정하고 꺼내는 프로그람을 호출하기 위한 지령을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CFLAGS_DEBUG

이 변수는 오유수정방식에서 C콤파일러용기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CFLAGS_MT

이 변수는 다중스레드응용프로그람을 창조하거나 혹은 련결하는 Qt의 판이 다중스레드 정적련결서고일 때 콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CFLAGS_MT_DBG

이 변수는 오유수정가능한 다중스레드응용프로그람을 창조하거나 련결하는 Qt의 판이 오유수정가능한 다중스레드 정적련결서고일 때 콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CFLAGS_MT_DLL

이것은 Windows에서만 사용된다

이 변수는 다중스레드 dll을 창조하거나 련결하는 Qt의 판이 다중스레드 dll일 때의 콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CFLAGS_MT_DLLDBG

이것은 Windows에서만 사용된다

이 변수는 오유수정가능한 다중스레드 dll을 창조하거나 련결하는 Qt의 판이 오유수정가능한 다중스레드 정적련결서고인 경우에 콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가

있다.

QMAKE_CFLAGS_RELEASE

이 변수는 오유수정불가능한 응용프로그람을 창조하기 위한 콤파일러기발을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CFLAGS_SHLIB

이것은 Unix가동환경에서만 사용된다

이 변수는 공유서고를 창조하기 위한 콤파일러기발을 포함한다 . 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CFLAGS_THREAD

이 변수는 다중스레드응용프로그람을 창조하기 위한 콤파일러기발을 포함한다 . 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CFLAGS_WARN_OFF

이 변수는 warn_off TEMPLATE선택이 지정되면 비지 않는다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CFLAGS_WARN_ON

이 변수는 warn_on TEMPLATE선택이 지정되면 비지 않는다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CLEAN

이 변수는 moc와 uic가 생성한 파일이 아닌 파일들과 "make clean"을 사용할 때 삭제되여야 하는 목적파일들을 포함한다.

QMAKE_CXXFLAGS_DEBUG

이 변수는 오유수정가능한 응용프로그람을 창조하기 위한 C++콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CXXFLAGS_MT

이 변수는 다중스레드응용프로그람을 창조하기 위한 C++콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CXXFLAGS_MT_DBG

이 변수는 오유수정가능한 다중스레드응용프로그람을 창조하기 위한 C++콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CXXFLAGS_MT_DLL

이것은 Windows에서만 사용된다

이 변수는 다중스레드 dll을 창조하기 위한 C++콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CXXFLAGS_MT_DLLDBG

이것은 Windows에서만 사용된다

이 변수는 오유수정가능한 다중스레드dll을 창조하기 위한 C++콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CXXFLAGS_RELEASE

이 변수는 응용프로그람을 창조하기 위한 C++콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CXXFLAGS_SHLIB

이 변수는 공유서고를 창조하기 위한 C++콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CXXFLAGS_THREAD

이 변수는 다중스레드응용프로그람을 창조하기 위한 C++콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CXXFLAGS_WARN_OFF

이 변수는 콤파일러경고를 금지하기 위한 C++콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_CXXFLAGS_WARN_ON

이 변수는 콤파일러경고를 생성하기 위한 C++ 콤파일러기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_EXTENSION_SHLIB

이 변수는 공유서고용확장자를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_FAILED_REQUIREMENTS

이 변수는 qmake를 리용하였을 때 부닥치게 되는 실패한 요구들의 목록을 포함한다. 실례로 sql모듈이 요구되는데 Qt로 콤파일되지 않는다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_FILETAGS

이 변수는 makefile에 입구되여야 할 SOURCES와 HEADERS와 같은 파일꼬리표들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_INCDIR

이 변수는 응용프로그람을 건설할 때 INCLUDEPATH에 추가되여야 할 알려진 모든 머리부파일들의 위치를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

POST_TARGETDEPS

목표가 의존하는 모든 서고들을 이 변수안에 렬거할수 있다. 일부는 이것을 유지하지 않는다. 여기에는 MSVC Dsp와 ProjectBuilder .pbproj파일들이 포함된다. 일반적으로 이것은 건설도구들에 의해 내적으로 유지되며 의존하는 정적서고들을 명시적으로 렬거하는데 쓸모있다.

이 목록은 모든 기본 (및 $$PRE_TARGETDEPS) 의존관계들의 다음에 놓인다.

PRE_TARGETDEPS

목표가 의존하는 모든 서고들을 이 변수안에 렬거할수 있다. 일부는 이것을 유지하지 않는다. 여기에는 MSVC Dsp와 ProjectBuilder .pbproj파일들이 포함된다. 일반적으로 이것은 건설도구들에 의해 내적으로 유지되며 의존하는 정적서고들을 명시적으로 렬거하는데 쓸모있다.

이 목록은 모든 기본 (및 $$PRE_TARGETDEPS) 의존관계들의 앞에 놓인다.

QMAKE_INCDIR_OPENGL

이 변수는 OpenGL을 유지하는 응용프로그람을 건설할 때 INCLUDEPATH에 추가되여야 할 OpenGL머리부파일들의 위치를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_INCDIR_QT

이 변수는 Qt응용프로그람을 건설할 때 INCLUDEPATH에 추가되여야 할 알려진 모든 머리부파일경로들의 위치를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_INCDIR_THREAD

이 변수는 다중스레드응용프로그람을 건설할 때 INCLUDEPATH에 추가되여야 할 알려진 모든 머리부파일경로들의 위치를 포함한다. 이 변수의 값은 전형적으로

qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_INCDIR_X11

이것은 Unix가동환경에서만 사용된다

이 변수는 X11응용프로그람을 건설할 때 INCLUDEPATH에 추가되여야 할 X11 머리부파일경로의 위치를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_CONSOLE

이것은 Windows에서만 사용된다

이 변수는 콘솔프로그람을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_CONSOLE_DLL

이것은 Windows에서만 사용된다

이 변수는 콘솔dll들을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_DEBUG

이 변수는 오유수정가능한 응용프로그람들을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_PLUGIN

이 변수는 플라그인들을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_QT_DLL

이 변수는 dll로서 건설한 Qt서고를 사용하는 프로그람들을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_RELEASE

이 변수는 출하용응용프로그람들을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_SHAPP

이 변수는 'app'형판을 사용하고있는 응용프로그람들을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_SHLIB

이 변수는 공유서고들을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_SONAME

이 변수는 .so 혹은 .dll와 같은 공유목적파일들의 이름을 설정하기 위한 련결기발들을 지정한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_THREAD

이 변수는 다중스레드프로젝트들을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_WINDOWS

이것은 Windows에서만 사용된다

이 변수는 windows프로젝트들을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LFLAGS_WINDOWS_DLL

이것은 Windows에서만 사용된다

이 변수는 windows dll프로젝트들을 건설할 때 련결기발들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBDIR

이 변수는 알려진 모든 서고등록부들의 위치를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBDIR_FLAGS

이것은 Unix가동환경에서만 사용된다

이 변수는 -L뒤붙이를 가지는 모든 서고등록부들의 위치를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

VPATH

이 변수는 qmake에게 열수 없는 파일들을 어디서 찾겠는가를 전한다. 이것을 리용하여 qmake에게 SOURCES와 같은것을 어디서 찾겠는가를 전달하고 SOURCES에서 열수 없는 항목을 찾으면 전체 VPATH목록을 조사하여 자체로 파일을 찾는다.

또한 DEPENDPATH를 참고.

DEPENDPATH

이 변수는 의존관계들을 해결하기 위해 조사해야 할 모든 등록부들의 목록을 포함한다. 이것은 'included'파일들을 조사할 때 사용된다.

QMAKE_LIBDIR_OPENGL

이 변수는 OpenGL서고등록부의 위치를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBDIR_QT

이 변수는 Qt서고등록부의 위치을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBDIR_X11

이것은 Unix가동환경에서만 사용된다

이 변수는 X11서고등록부의 위치을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS

이 변수는 모든 프로젝트서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_CONSOLE

이것은 Windows에서만 사용된다

이 변수는 콘솔응용프로그람을 건설할 때 련결되여야 할 모든 프로젝트서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_OPENGL

이 변수는 모든 OpenGL서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_OPENGL_QT

이 변수는 모든 OpenGL Qt서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_QT

이 변수는 모든 Qt서고들을 포함한다.이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_QT_DLL

이것은 Windows에서만 사용된다

이 변수는 Qt가 dll로서 건설될 때 Qt서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_QT_OPENGL

이 변수는 OpenGL유지가 선택되였으면 련결해야 할 모든 서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_QT_THREAD

이 변수는 스레드유지가 선택되였을 때 련결해야 할 모든 서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_RT

이것은 Borland콤파일러에서만 사용된다

이 변수는 응용프로그람을 건설할 때 련결해야 할 실시간서고를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_RTMT

이것은 Borland콤파일러에서만 사용된다

이 변수는 다중스레드응용프로그람을 건설할 때 련결해야 할 실행시서고를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_THREAD

이것은 Unix가동환경에서만 사용된다

이 변수는 다중스레드응용프로그람을 건설할 때 련결해야 할 모든 서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_WINDOWS

이것은 Windows에서만 사용된다

이 변수는 모든 windows서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_X11

이것은 Unix가동환경에서만 사용된다

이 변수는 모든 X11서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIBS_X11SM

이것은 Unix가동환경에서만 사용된다

이 변수는 모든 X11쎄숀관리서고들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LIB_FLAG

이 변수는 'lib'형판이 지정되면 비지 않는다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_LINK_SHLIB_CMD

이 변수는 공유서고를 창조할 때 실행해야 할 지령을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_POST_LINK

이 변수는 TARGET를 모두 련결한 후에 실행할 지령을 포함한다. 이 변수는 보통 비여있으므로 아무것도 실행되지 않으며 또한 일부는 이것을 유지하지 않고 대체로 Makefile뒤면(backend)처리만 유지한다.

QMAKE_PRE_LINK

이 변수는 TARGET를 모두 련결하기전에 실행할 지령을 포함한다. 이 변수는 보통 비여있으므로 아무것도 실행되지 않으며 또한 일부는 이것을 유지하지 않고 대체로 Makefile뒤면(backend)처리만 유지한다.

QMAKE_LN_SHLIB

이 변수는 공유서고에로의 련결을 창조할 때 실행할 지령을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_MAKEFILE

이 변수는 창조할 makefile의 이름을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_MOC_SRC

이 변수는 프로젝트에 생성 및 포함하여야 할 모든 moc원천파일들을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_QMAKE

이 변수는 qmake가 경로안에 없을 때 그 위치를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_QT_DLL

이 변수는 Qt가 dll로서 건설되였다면 비지 않는다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_RUN_CC

이 변수는 객체를 건설하는데 필요한 개별적인 규칙들을 지정한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_RUN_CC_IMP

이 변수는 객체를 건설하는데 필요한 개별적인 규칙들을 지정한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_RUN_CXX

이 변수는 객체를 건설하는데 필요한 개별적인 규칙들을 지정한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_RUN_CXX_IMP

이 변수는 객체를 건설하는데 필요한 개별적인 규칙들을 지정한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_TARGET

이 변수는 프로젝트목표의 이름을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

QMAKE_UIC

이 변수는 uic가 경로안에 없는 경우 그 위치를 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

물론 uic에 추가적인 플라그인경로들과 같은 인수들을 지정하는데 쓰일수 있다. 실례로

```
QMAKE_UIC = uic -L /path/to/plugin
```

RC_FILE

이 변수는 응용프로그람용자원파일의 이름을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

RES_FILE

이 변수는 응용프로그용자원파일의 이름을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

SRCMOC

이 변수는 Q_OBJECT마크로를 포함하는 파일들을 발견할수 있는 경우에 qmake.에 의해 설정된다. SRCMOC는 생성된 모든 moc파일들의 이름을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

TARGET_EXT

이 변수는 목표의 확장자를 지정한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

TARGET_x

이 변수는 기본판번호를 가지는 목표의 확장자를 지정한다. 이 변수의 값은 전형적

으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

TARGET_x.y.z

이 변수는 판번호를 가지는 목표의 확장자를 지정한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

UICIMPLS

이 변수는 UIC가 생성한 실현파일들의 목록을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

UICOBJECTS

이 변수는 UICIMPLS변수로부터 생성된다. 매개 파일의 확장자는 .o (Unix) 혹은 .obj (Win32)로 교체된다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

VER_MAJ

이 변수는 'lib' 형판이 지정되면 서고의 기본판번호를 포함한다.

VER_MIN

이 변수는 'lib' 형판이 지정되면 서고의 보조판번호를 포함한다.

VER_PAT

이 변수는 'lib' 형판이 지정되면 서고의 패치(patch)판번호를 포함한다.

QMAKE_EXT_MOC

이 변수는 포함된 moc파일들에서 사용한 확장자를 변경한다.

또한 파일확장자참고.

QMAKE_EXT_UI

이 변수는 /e Designer UI파일들에 사용한 확장자를 변경한다.

QMAKE_EXT_PRL

이 변수는 창조한 PRL파일들에 사용된 확장자를 변경한다.

QMAKE_EXT_LEX

이 변수는 lex에 주어진 파일들에 사용한 확장자를 변경한다.

QMAKE_EXT_YACC

이 변수는 yacc에 주어진 파일들에 사용한 확장자를 변경한다.

QMAKE_EXT_OBJ

이 변수는 생성한 목적파일들에 사용한 확장자를 변경한다.

QMAKE_EXT_CPP

이 변수는 이 값목록안의 모든 뒤붙이들을 C++원천코드형의 파일들로 해석한다.

QMAKE_EXT_H

이 변수는 이 값목록안의 모든 뒤붙이들을 C머리부파일형의 파일들로 해석한다.

YACCIMPLS

이 변수는 yacc원천파일들의 목록을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

YACCOBJECTS

이 변수는 yacc목적파일들의 목록을 포함한다. 이 변수의 값은 전형적으로 qmake 혹은 qmake.conf에 의하여 조종되며 드문히 수정할 필요가 있다.

3. 함수

qmake는 다음과 같은 함수들을 인식한다.

include(filename)

이 함수는 filename의 내용을 현재 프로젝트안의 포함될 위치에 포함한다. 함수는 filename이 포함되면 성공하고 그렇지 않으면 실패한다. 유효범위를 사용하여 이 함수의 돌림값을 검사할수 있다.

실례

```
include( shared.pri )
OPTIONS = standard custom
!include( options.pri ) {
    message( "No custom build options specified" )
    OPTIONS -= custom
}
```

exists(file)

이 함수는 file이 존재하는가 검사한다. 파일이 존재하면 성공하고 그렇지 않으면 실패한다. 파일에 정규식을 지정할수 있으며 임의의 파일이 지정된 정규식과 대조되면 성공한다.

실례

```
exists( $(QTDIR)/lib/libqt-mt* ) {
    message( "Configuring for multi-threaded Qt..." )
    CONFIG += thread
}
```

contains(variablename, value)

이 함수는 변수 variablename가 값 value를 포함하면 성공한다. 유효범위를 사용하여 이 함수의 돌림값을 검사할수 있다.

실례

```
contains( drivers, network ) {
    # drivers contains 'network'
```

```
    message( "Configuring for network build..." )
    HEADERS += network.h
    SOURCES += network.cpp
}
```

count(variablename, number)

이 함수는 변수 variablename가 number요소를 포함하면 성공하고 그렇지 않으면 실패한다. 유효범위를 사용하여 이 함수의 돌림값을 검사할수 있다.

실례

```
MYVAR = one two three
count( MYVAR, 3 ) {
    # always true
}
```

infile(filename, var, val)

이 함수는 파일 filename(qmake자체에 의해 해석될 때)이 값 val을 가진 변수 var를 포함하면 성공한다. 또한 제3인수 (val)를 넘기지 않을수 있으며 함수는 var 가 파일안에 할당되였는가만 검사한다.

isEmpty(variablename)

이 함수는 변수 variablename이 비였으면 (count(variable, 0)이면) 성공한다.

system(command)

이 함수는 둘째쉘에서 command을 실행하며 지령이 완료상태 1로 끝나면 성공한다. 유효범위를 사용하여 이 함수의 돌림값을 검사할수 있다.

실례

```
system(ls /bin):HAS_BIN=FALSE
```

message(string)

이 함수는 늘 성공하며 사용자에게 주어진 string을 현시한다.

error(string)

이 함수는 절대로 값을 돌려주지 않는다. 사용자에게 주어진 string을 현시하고 qmake를 완료한다. 이 함수는 오직 매우 중요한 환경구성에만 사용되여야 한다.

실례

```
release:debug:error(You can't have release and debug at the same time!)
```

4. 속성

qmake는 영속정보체계를 가지며 이것은 qmake에서 일단 변수를《설정》하게 하며 qmake가 호출될 때마다 이 값이 질문될수 있다. 다음것을 사용하여 qmake에서 속성을 설정한다.

qmake -set VARIABLE VALUE

qmake로부터 이 정보를 얻으려면 다음과 같이 할수 있다.

qmake -query VARIABLE

qmake -query #queries all current VARIABLE/VALUE pairs..

이 정보는 QSettings객체에 보관된다.(이것은 서로 다른 가동환경들에서 서로 다른곳에 보관된다.) VARIABLE이 판으로 될 때 낡은 판의 qmake에 한개 값을 설정할수 있으며 새로운 판들이 이 값을 얻지만 qmake의 새 판에 VARIABLE을 설정하면 낡은 판이 이 값을 사용하지 않는다. 그러나 다음과 같이 qmake의 판을 VARIABLE앞에 놓으면 변수의 특정판을 질문할수 있다.

qmake -query "1.06a/VARIABLE"

또한 qmake는 'builtin'속성들의 표기를 가진다. 실례로 QT_INSTALL_PREFIX속성을 사용하여 이 판의 qmake에 대하여 Qt의 설치를 질문할수 있다.

qmake -query "QT_INSTALL_PREFIX"

이 기본속성들이 판이 아닐 때 그것들앞에 판을 놓을수 없으며 매개 qmake는 이 값들의 자체의 표기를 가진다. 아래의 목록은 기본속성들을 보여준다.

- QT_INSTALL_PREFIX - qmake가 상주용으로 건설되는 Qt판의 위치
- QT_INSTALL_DATA - 이 판의 Qt자료가 상주하는곳
- QMAKE_VERSION - qmake의 현재판

끝으로 이 값들은 다음과 같이 특정한 표기를 가지는 프로젝트파일안에서 질문될수 있다.

QMAKE_VERS = $$[QMAKE_VERSION]

5. 환경변수와 환경구성

1) QMAKESPEC

qmake는 적당한 makefile들을 생성하는데 쓰이는 수많은 기정값들을 포함하는 가동환경과 콤파일러서술파일을 요구한다. 표준Qt배포물은 이러한 파일들을 수많이 포함하며 그것들은 Qt설치의 'mkspecs'보조등록부에 배치된다.

QMAKESPEC환경변수는 다음의 값들중 임의의것을 포함할수 있다.

- qmake.conf파일을 포함하는 등록부에로의 완전경로. 이 경우에 qmake는 그 등록부안으로부터 qmake.conf파일을 연다. 파일이 존재하지 않으면 qmake는 오유로 완료한다.
- 가동환경-콤파일러결합의 이름. 이 경우에 qmake는 QTDIR환경변수에 의해 지정된 등록부를 탐색한다.

알아두기: QMAKESPEC경로는 INCLUDEPATH체계변수에 자동적으로 추가된

다.

2) INSTALLS

건설할 때(실례로 make install) 같은 편의프로그람으로부터 설치할수 있게 하는것이 UNIX에서 상식이다. 이를 위하여 qmake는 설치모임의 개념을 도입하였다. 이를 위한 표기는 아주 간단하며 우선 qmake실례에서 《객체》를 채운다.

```
documentation.path = /usr/local/program/doc
documentation.files = docs/*
```

이 방법에서 qmake에게 이 설치에 대하여 몇가지 말하고있다. 첫째로 /usr/local/program/doc(path성원)에 설치하려고 계획하며 둘째로 docs등록부에 모든것을 복사할 계획이다. 이것을 수행하면 설치목록에 그것을 삽입할수 있다.

```
INSTALLS += documentation
```

그러면 qmake는 정확한것들이 지정된곳에 복사되는가 확인한다. 그러나 더 큰 조종을 요구한다면 객체의 'extra'성원을 요구할수 있다.

```
unix:documentation.extra = create_docs; mv master.doc toc.doc
```

그다음 qmake는 나머지를 실행한다. (이것은 물론 가동환경에 고유하므로 우선 자기 가동환경에서 시험할 필요가 있으며 이 경우에는 unix에서 시험한다.) 그때 파일성원의 표준처리를 수행한다. 끝으로 INSTALLS에 기본설치를 추가한다면 qmake(그리고 파일이나 여분의 성원을 지정하지 않는다)는 무엇을 복사해야 하는가를 결심하며 현재는 오직 유지된 기본설치만 목표이다.

```
target.path = /usr/local/myprogram
INSTALLS += target
```

이리하여 qmake는 무엇을 복사해야 하는가 알고있으며 이것을 수행한다.

3) 캐쉬파일

캐쉬파일(우의 선택에서 언급)은 qmake가 qmake.conf파일, .pro파일 혹은 지령행에서 지정되지 않은 설정값들을 찾기 위하여 읽어들이는 특수파일이다. -nocache이 지정되지 않으면 qmake는 부모등록부들에서 .qmake.cache 라는 파일을 찾으려고 한다. 이 파일을 찾는데서 실패하면 이 처리단계를 말없이 무시한다.

4) 서고의존관계

흔히 서고에 련결할 때 qmake는 기초하고있는 가동환경에 의거하여 이 서고를 어떤 다른 서고와 련결해야 하는가를 알아내고 가동환경이 그것들을 포함하게 한다. 그러나 많은 경우에 이것은 충분하지 않다. 실례로 서고를 정적으로 련결할 때 련결할 서고가 없으므로 그 서고들에 대한 의존관계들이 창조된다. 그러나 후에 이 서고에 련결하는 응용프로그람은 서고에서 련결되는것들이 요구하는 기호들을 찾기 위한 위치를 알아야 한다. 이러한 상황을 방조하기 위하여 qmake는 적당하다고 느껴질 때 서고의 의존관계

를 따르지만 이 동작은 qmake에서 허용되여야 한다. 그러자면 두 단계가 요구된다. 우선 서고에서 그것을 허용해야 한다. 그러자면 qmake에게 이 서고에 대한 정보를 보관할것을 알려야 한다.

CONFIG += create_prl

이것은 오직 lib형판에만 관련되며 다른 모든것은 무시된다. 이 선택이 허용될 때 qmake는 서고에 대한 메타정보를 보관하는 파일(.prl파일)을 창조한다. 이 메타파일은 그자체가 바로 qmake프로젝트파일이지만 모든 내부변수들을 가진다. 이 파일을 보는것은 자유이다. 지워졌으면 qmake는 필요할 때(.pro파일을 후에 읽어들일 때 혹은 아래에 보여주는것처럼 의존서고가 변경된 경우에) 다시 창조하는것을 알고있다. 우에서 INSTALLS안의 목표를 사용하여 이 서고를 설치할 때 qmake는 .prl파일을 자기의 설치경로에 자동적으로 복사한다.

이 처리를 허용하는 둘째 단계는 우에서 창조된 메타정보의 읽기를 선택하는것이다.

CONFIG += link_prl

이것이 설정될 때 qmake는 련결된 서고들을 모두 처리하고 그 메타정보를 찾는다. 이 메타정보를 리용하여 qmake는 련결과 관계되는것들을 종합하고 특히 LIBS는 물론 자기의 DEFINES의 목록에 추가한다. 일단 qmake가 이 파일을 처리하였으면 새로 생겨난 LIBS들을 조사하여 그 의존하는 .prl파일들을 찾고 모든 서고들을 해결하였을 때까지 계속한다. 이 시점에서 보통 makefile이 창조되고 서고들은 명시적으로 자기 프로그람에 련결된다.

.prl파일의 내부는 닫겨져있으므로 후에 쉽게 변경할수 있다. 이것은 손으로 변경할수 있게 설계되지 않지만 qmake에 의해서만 창조되여야 한다. 이 .prl파일들은 또한 가동환경에 의존할 때(makefile처럼) 조작체계사이를 옮기지 말아야 한다.

5) 파일확장자

보통환경에서 qmake는 자기의 가동환경을 위한 적당한 파일확장자를 사용하려고 한다. 여러가지 경우가 있을수 있지만 보통 이 확장자들을 무시하려고 한다. 그러자면 자기의 .pro파일에서 기본(builtin)변수들을 수정해야 하며 그 변수들은 또한 이 파일들의 qmake해석을 변경한다. 다음과 같이 수행할수 있다.

QMAKE_EXT_MOC = .mymoc

변수들은 다음과 같다.

QMAKE_EXT_MOC - 포함한 moc파일들에 배치된 확장자를 수정한다.

QMAKE_EXT_UI - designer UI파일들(보통 FORMS에서)에 사용된 확장자를 수정한다.

QMAKE_EXT_PRL - 서고의존관계파일들에 배치된 확장자를 수정한다.

QMAKE_EXT_LEX -파일들에 사용된 뒤붙이를 변경한다.(보통 LEXSOURCES에서).

QMAKE_EXT_YACC - 파일들에 사용된 뒤붙이를 변경한다. (보통

YACCSOURCES에서).

QMAKE_EXT_OBJ - 생성된 목적파일들에 사용된 뒤붙이를 변경한다.

우에서는 바로 첫째값을 받아들이므로 거기에 자기의 makefile에 사용해야 할 값을 할당해야 한다. 값목록을 받아들이는 두개의 변수들이 있는데 그것들은 다음과 같다.

- QMAKE_EXT_CPP - 해석에서 이 뒤붙이를 가지는 모든 파일들을 C++ 원천파일들로 변경한다.
- QMAKE_EXT_H - 해석에서 이 뒤붙이를 가지는 모든 파일들을 C머리부파일들로 변경한다.

6) Makefile출력의 전용화

흔히 qmake는 모든것을 건설도구를 리용하여 수행하려고 하는데 이것은 특수한 가동환경에 의존하는 지령들을 실제로 실행할 필요가 있을 때 리상적이지 못하다. 이것은 각이한 qmake뒤면처리를 위한 특정한 지령에 의해 이룩될수 있다. (현재 이것은 UNIX 생성기에서만 유지하고있다.)

Makefile을 전용화하기 위한 대면부는 qmake안의 다른곳에서처럼 《객체》들을 통하여 수행된다. 이를 위한 표기는 아주 간단하며 우선 qmake실례의 《객체》를 채운다.

```
mytarget.target = .buildfile
mytarget.commands = touch $$mytarget.target
mytarget.depends = mytarget2
mytarget2.commands = @echo Building $$mytarget.target
```

우의 정보는 .buildfile이라는 Makefile목표를 포함하는 mytarget라는 qmake목표를 정의하며 .buildfile은 'touch .buildfile'에 의해 생성되고 끝으로 이 Makefile 목표는 qmake목표 mytarget2에 의존한다. 추가적으로 stdout에 무엇인가 단순히 반영하는 qmake목표mytarget2를 정의하였다.

우의것을 사용하게 하는 마지막 단계는 실제로 이것이 qmake의 목표건설부분들에 의해 다음과 같이 사용되는 객체라는것을 qmake에 지령하는것이다:

```
QMAKE_EXTRA_UNIX_TARGETS += mytarget mytarget2
```

이것이 qmake로 전용목표들을 실제로 건설할 때 수행해야 하는 모든것이다. 물론 목표들중 하나를 종속시켜 qmake건설목표를 실제로 건설하려고 할수 있다. 그러자면 PRE_TARGETDEPS목록에 자기의 Makefile목표를 단순히 포함할 필요가 있다.

편리상 새로운 범용콤파일러(혹은 지어 앞처리기)용으로 (UNIX)프로젝트들을 전용화하는 방법도 있다.

```
new_moc.output  = moc_${QMAKE_FILE_BASE}.cpp
new_moc.commands = moc ${QMAKE_FILE_NAME} -o ${QMAKE_FILE_OUT}
new_moc.depends = g++ -E -M ${QMAKE_FILE_NAME} | sed "s,^.*: ,,"
```

```
new_moc.input  =  NEW_HEADERS
QMAKE_EXTRA_UNIX_COMPILERS  +=  new_moc
```

이것을 리용하여 qmake용의 새로운 moc를 창조할수 있으며 지령들은 NEW_HEADERS변수(입력변수로부터)에 주어진 모든 인수들에 대하여 실행되고 출력에 써넣는다. 그리고 자기 목표에 련결하기 위하여 이 파일이름을 콤파일러에 자동적으로 넘긴다. 또한 qmake가 실행하는것은 의존관계정보의 생성에 의존하며 물론 프로젝트파일에 이것을 배치한다.

이 지령들은 캐쉬파일에 쉽게 배치될수 있으며 그 다음의 .pro파일들은 NEW_HEADERS에 여러개의 인수를 줄수 있다.

제5장. 기타 도구들

제1절. QEmbed - 파일 및 화상매몰프로그람

QEmbed도구는 qt/tools/qembed에 있으며 임의의 파일을 C++코드로 변환한다. 이것은 외부파일들로부터 자료를 적재하지 않고 화상파일들과 다른 자원들을 직접 포함하는데 쓸모있다.

또한 QEmbed는 자기의 응용프로그람에 직접 포함할수 있는 비압축판의 화상들을 생성하여 외부파일과 화상파일형식해석요구를 피할수 있다. 이것은 압축이 큰 리득을 주지 않는 그림기호과 같은 작은 화상들에 사용할수 있다.

1. 사용법

qembed [general-files] [--images image-files]

·general-files

이 파일들은 임의의 형의 파일일수 있다.

·--images image-files

이 파일들은 Qt가 유지하는 화상형식이여야 한다.

2. 출력

QEmbed로부터의 출력은 C++원천파일에서 포함해야 할 C++머리부파일이다. 원천파일에서 자기의 응용프로그람에 적합한 래퍼함수를 만들어야 한다. 두 함수가 제공된다. 래퍼함수는 이것들중 하나를 호출하거나 자체로 실현할수 있다. 여기에 제공된 매개 함수들을 사용하는 간단한 실례가 있다.

① qembed_findImage()

```
#include "generated_qembed_file.h"
QImage myFindImage(const char* name)
{
  return qembed_findImage(name);
}
```

생성된 함수를 호출한다. name은 확장자없는 원시화상파일이름이다.

② qembed_findData()

```
#include "generated_qembed_file.h"
QByteArray myFindData(const char* name)
{
```

```
return  qembed_findData(name);
}
```

생성된 함수를 호출한다. name은 확장자있는 원시화상파일이름이다.

또한 QEmbed로부터의 출력을 고찰하고 자기의 요구에 맞는 함수를 쓴다.

제2절. Qt/Embedded가상틀완충기

가상틀완충기는 탁상기계에서 콘솔과 X11을 절환하지 않고 Qt/Embedded프로그람들을 개발하게 한다.

가상틀완충기는 $QTDIR/tools/qvfb에 위치되여있다.

1. 가상틀완충기의 사용

① -qvfb인수를 가지고 Qt/Embedded의 환경을 구성하고 서고를 콤파일한다.

```
./configure -qvfb
make
```

② 표준Qt/X11응용프로그람으로서 qvfb를 콤파일하고 실행한다. Qt/Embedded응용프로그람로서 그것을 콤파일하여서는 안된다.

③ Qt/Embedded주응용프로그람을 기동한다. 즉 QApplication::GuiServer기발을 리용하여 QApplication을 구축하거나 -qws지령행파라메터를 사용한다. 가상틀완충기구동기를 사용하려고 한다는것을 봉사기에게 지정할수 있다. 실례로

```
masterapp -qws -display QVFb:0
```

④ Qt/Embedded는 qvfb를 자동탐지하므로 qvfb를 실행하고있고 Qt/Embedded서고가 qvfb를 유지한다는것을 알고있으면 -display지령행파라메터를 생략할수 있다. (그렇지 않으면 Qt/Embedded는 실제의 틀완충기에 써넣으며 자기의 X11현시기는 제대로 동작하지 않는다.)

qvfb는 다음과 같은 지령행선택을 유지한다.

표 5-1. qvfb의 지령행선택

선택	의미
-width width	가상틀완충기의 폭 (기정값: 240).
-height height	가상틀완충기의 높이 (기정값: 320).
-depth depth	가상틀완충기의 깊이 (1, 8 or 32; 기정값: 8).
-nocursor	틀완충기창문에서 X11유표를 표시하지 않는다.
-qwsdisplay :id	제공하려는 Qt/Embedded현시기식별자 (기정값: :0).

2. 가상틀완충기설계

가상틀완충기는 공유기억령역(가상틀완충기)과 틀완충기를 창문에 현시하는 편의프로그람(qvfb)을 리용하여 틀완충기를 모의한다. 변경된 현시기의 령역은 주기적으로 갱신되므로 매개의 개별적인 그리기조작보다 리산적인 틀완충기의 장면들을 보게 된다. 이러한 리유로 깜빡거림과 같은 그리기문제는 프로그람이 실제의 틀완충기를 리용하여 실행될 때까지 결에 나타나지 않는다.

목표재생비률은 View|Refresh Rate차림표항목을 거쳐서 설정될수 있다. 이것은 qvfb가 갱신된 령역을 더 자주 검사하게 한다. 그 비률은 목표뿐이다. 약간한 그리기가 수행되고있으면 틀완충기는 그리기사건들사이의 갱신을 표시하지 않는다. 응용프로그람이 동화상을 표시하고있으면 갱신은 빈번해지고 응용프로그람과 qvfb는 처리소자시간에 완료한다.

마우스와 건반사건들은 이름있는 파이프를 거쳐서 Qt/Embedded마스터처리에 넘겨진다.

가상틀완충기는 개발도구뿐이다. 보안문제는 가상틀완충기설계에서 고려되지 않으며 산품환경에서 피해야 하며 -qvfb를 리용하여 산품서고들의 환경을 구성하지 않는다.

제3절. makeqpf

서체들을 묘사하고 보관함으로써 QPF서체파일들을 절약한다.

사용법

makeqpf [-A] [-f spec-file] [font ...]

-A - fontdir에 모든 서체들을 묘사하고 보관한다.

-f - 행들의 파일: 서체이름 문자-범위 실례로 "smoothtimes 0-ff,20a0-20af"

font - 묘사하고 보관하려는 서체

제4절. 메타객체콤파일러 moc

메타객체콤파일러(Meta Object Compiler) moc는Qt의 C++확장을 조종하는 프로그람이다.

moc는 C++원천파일을 읽어들인다. 그것이 Q_OBJECT마크로를 포함하는 하나이상의 클라스선언을 발견하면 Q_OBJECT마크로를 사용하는 클라스들에 대한 메타객체코드를 포함하는 다른 C++원천파일을 생성한다. 다른것들중에서 메타객체코드는 신호-처리부기구, 실행시형정보 및 동적속성체계에 요구된다.

moc가 생성한 C++원천파일은 클라스의 실현과 함께 콤파일되고 련결되여야 한다. 혹은 클라스의 원천파일에 포함(include)될수 있다.

qmake를 사용하여 자기의 Makefile들을 창조한다면 필요할 때 moc를 호출하는 건설규칙들이 포함되므로 moc를 직접 사용할 필요가 없다. moc에 대한 더 자세한 정보는 《Qt일반지식》 3장 6절을 참고하시오.

1. 사용법

moc는 보통 다음과 같이 클라스선언을 포함하는 입력파일에서 사용된다.

```
class MyClass : public QObject
{
  Q_OBJECT
public:
  MyClass( QObject * parent=0, const char * name=0 );
  ~MyClass();
signals:
  void mySignal();
public slots:
  void mySlot();
};
```

우에서 보여준 신호 및 처리부와 함께 moc는 또한 다음 실례에서처럼 객체속성들을 실현한다. Q_PROPERTY마크로는 객체속성을 선언하며 한편 Q_ENUMS은 속성체계안에서 사용할수 있도록 클라스안에 렬거형목록을 선언한다. 이와 같이 특수한 경우에 priority라고도 불리우며 하나의 얻기함수 priority()와 하나의 설정함수 setPriority()를 가지는 렬거형 Priority의 속성을 선언한다.

```
class MyClass : public QObject
{
  Q_OBJECT
  Q_PROPERTY( Priority priority READ priority WRITE setPriority )
  Q_ENUMS( Priority )
public:
  MyClass( QObject * parent=0, const char * name=0 );
  ~MyClass();

  enum Priority { High, Low, VeryHigh, VeryLow };
  void setPriority( Priority );
```

```
    Priority priority() const;
  };
```

속성들은 Q_OVERRIDE마크로에 의해 파생클라스에서 수정될수 있다. Q_SETS마크로는 모임으로서 즉 모두 OR될수 있는 enum들을 선언한다. 다른 마크로 Q_CLASSINFO는 이름-값 쌍을 클라스의 메타객체에 련결하는데 쓰일수 있다.

```
  class MyClass : public QObject
  {
    Q_OBJECT
    Q_CLASSINFO( "Author", "Oscar Peterson")
    Q_CLASSINFO( "Status", "Active")
  public:
    MyClass( QObject * parent=0, const char * name=0 );
    ~MyClass();
  };
```

3가지 개념 즉 신호 및 처리부, 속성과 클라스메타자료를 결합할수 있다.

moc에 의해 생성된 출력은 자기의 프로그람에서 다른 C++코드와 같이 콤파일되고 련결되여야 한다. 그렇지 않으면 건설은 최종련결단계에서 실패한다. 편리상 이것은 다음의 두가지 방법들중 하나에 의해 수행된다.

방법1: 클라스선언을 머리부파일(.h)에서 발견한다.

클라스선언이 파일 myclass.h에서 발견되면 moc출력은 moc_myclass.cpp라는 파일에 놓여야 한다. 그때 이 파일은 보통과 같이 콤파일되여 목적파일 moc_myclass.o (Unix에서) 혹은 moc_myclass.obj (Windows에서)가 생성된다. 그때 이 목적파일은 프로그람의 최종건설단계에서 모두 련결되는 목적파일목록에 포함되여야 한다.

방법2: 클라스선언을 실현파일(.cpp)에서 발견한다.

클라스선언이 파일 myclass.cpp에 있다면 moc출력은 myclass.moc라는 파일에 놓여야 한다. 이 파일은 실현파일에 포함되여야 한다. 즉 myclass.cpp은 끝에 다음 행을 포함하여야 한다.

```
#include "myclass.moc"
```

이것은 moc가 생성한 코드를 콤파일하여 myclass.cpp안의 보통클라스정의와 련결되게 하므로 방법A에서처럼 그것을 따로따로 콤파일하고 련결할 필요가 없다.

방법A는 보통의 방법이다. 방법B는 실현파일이 자체로 포함되게 하려고 하는 경우나 Q_OBJECT클라스가 내부실현이므로 머리부파일에서 볼수 없는 경우에 사용될수 있다.

2. Makefile에 의하여 moc사용의 자동화

어떤 가장 단순한 시험프로그람에서는 moc의 실행을 자동화할것을 권고한다. 자기 프로그람의 Makefile에 규칙들을 추가함으로써 make는 필요할 때 moc의 실행과 moc 출력의 조종을 고려할수 있다.

Trolltech의 무료의 makefile생성도구인 qmake를 사용하여 자기의 Makefile들을 건설할것을 권고한다. 이 도구는 방법A와 B형식의 원천파일들을 모두 인식하며 필요한 moc조종을 모두 수행하는 Makefile을 생성한다.

Makefile들을 자체로 창조하려고 한다면 여기에 moc조종을 포함하는 방법에 대한 암시가 있다.

머리부파일안의 Q_OBJECT클라스선언에 대하여서는 아래에 GNU make를 사용하는 경우에만 쓸모있는 makefile이 있다.

```
moc_%.cpp: %.h
    moc $< -o $@
```

이식할수 있게 하려면 다음의 형식으로 독특한 규칙들을 사용할수 있다.

```
moc_NAME.cpp: NAME.h
    moc $< -o $@
```

또한 자기의 SOURCES(자기가 애호하는 이름을 대리)변수에 moc_NAME.cpp를, OBJECTS변수에 moc_NAME.o 혹은 moc_NAME.obj를 추가하는것을 기억하여야 한다.(C++원천파일들을 .cpp라고 이름짓기를 좋아하여도 moc가 그것을 고려하지 않으므로 .C, .cc, .CC, .cxx 혹은 .c++를 사용할수도 있다.)

실현파일(.cpp)의 Q_OBJECT클라스선언에서는 다음과 같은 makefile규칙을 제안한다.

```
NAME.o: NAME.moc
NAME.moc: NAME.cpp
    moc -i $< -o $@
```

이것은 NAME.cpp을 콤파일하기전에 make가 moc를 실행한다는것을 담보한다. 그다음 NAME.cpp의 끝에 다음 행을 넣을수 있다.

```
#include "NAME.moc"
```

그 파일에서 선언한 모든 클라스들은 충분히 알려져있다.

3. moc의 호출

여기에 moc에 의해 지원되는 지령행선택들이 있다.

표 5-2. moc의 지령행선택

선택	의미
-o file	출력을 stdout가 아니라 file에 써넣는다.
-f	출력에 #include문의 생성을 강요한다. 이것은 이름이 정규식 \.[hH][^.]* (즉 확장자는 H 혹은 h로 시작한다.)과 일치하는 파일들에 기정이다. 이 선택은 오직 표준명명관례를 따르지 않는 머리부파일이 있는 경우에 사용할수 있다.
-i	출력에서 #include문을 생성하지 않는다. 이것은 하나이상의 클라스선언을 포함하는 C++파일에 대하여 moc를 실행하는데 쓰일수 있다. 그때 .cpp파일에 메타객체코드를 포함하여야 한다. -i와 -f가 둘다 제시되면 -f가 선택된다.
-nw	어떤 경고도 생성하지 않는다. 권고하지 않는다.
-ldbg	stdout에 대량적인 문법오유수정정보를 써넣는다.
-p path	moc가 생성된 #include문(생성된 경우)안의 파일이름에로의 path/를 받아들이게 한다(prepend).
-q path	moc가 생성된 코드안의 qt #include파일들의 파일이름에로의 path/ 를 받아들이게 한다.

moc가 머리부파일부분을 구문해석하지 못한다는것을 명백히 말할수 있다. 이것은 부분문자렬 MOC_SKIP_BEGIN 혹은 MOC_SKIP_END를 포함하는 C++설명문(//)을 인식한다. 그것들은 기대한대로 작업하며 그것들의 여러 준위를 가질수 있다. moc로 보았을 때 순수한 결과는 MOC_SKIP_BEGIN과 MOC_SKIP_END사이의 모든 행들을 삭제한것과 같다.

4. 진단

moc는 Q_OBJECT클라스선언안의 많은 위험한 혹은 옳지 않은 구성에 대하여 경고한다.

자기 프로그람의 최종건설단계에서 YourClass::className()가 정의되지 않거나 YourClass가 vtbl를 잃었다는것을 말하는 련결오유들을 얻으면 일부는 틀리게 수행되였다. 대체로 moc가 생성한 C++코드를 콤파일하거나 포함(include)하는것을 잊었거나 (전자의 경우에) link지령에서 그 지령을 포함하는것을 잊었다.

5. 제한

moc는 #include 혹은 #define를 전개하지 못하며 단순히 앞처리지령을 만날 때마다 뛰여넘는다. 이것은 보통 실천에서는 문제가 없다.

moc는 C++의 모두를 조종하지 못한다. 기본문제는 클라스형판들이 신호나 처리부를 가질수 없는것이다. 여기에 실례가 있다:

```
class SomeTemplate<int> : public QFrame {
  Q_OBJECT
  ...
signals:
  void bugInMocDetected( int );
};
```

좀 중요하지만 다음의 구성은 비법적이다.

1) 다중계승은 QObject을 처음에 놓을것을 요구한다

다중계승을 사용하고있다면 moc는 처음으로 계승하는 클라스가 QObject의 파생클라스라고 가정한다. 또한 처음으로 계승된 클라스가 QObject라는것을 확인해야 한다.

```
class SomeClass : public QObject, public OtherClass {
  ...
};
```

(이러한 제한은 거의 제거할수 없다. moc가#include 혹은 #define을 전개하지 못하므로 어느 기초클라스가 QObject인가 찾을수 없다.)

2) 함수지적자들은 신호나 처리부에 인수로 될수 없다

신호-처리부인수로서 함수지적자의 사용을 고려하는 대부분의 경우에 계승이 더 좋은 후보라고 생각한다. 여기에 틀린 문법의 실례가 있다.

```
class SomeClass : public QObject {
  Q_OBJECT
  ...
public slots:
  // illegal
  void apply( void (*apply)(List *, void *), char * );
};
```

이 제한에 대하여 다음과 같이 작업할수 있다.

```
typedef void (*ApplyFunctionType)( List *, void * );


class SomeClass : public QObject {
  Q_OBJECT
  ...
public slots:
```

```
  void apply( ApplyFunctionType, char * );
 };
```

흔히 함수지적자를 계승과 가상함수, 신호나 처리부로 바꾸는것이 더 나을수 있다.

3) 동료선언은 신호나 처리부절에 놓일수 없다

흔히 이것은 동작하지만 일반적으로 동료선언은 신호나 처리부절에 배치될수 없다. 그대신에 그것을 private, protected 혹은 public절에 놓아야 한다. 여기에 옳지 않은 문법의 실례가 있다.

```
 class SomeClass : public QObject {
  Q_OBJECT
  ...
 signals:
  friend class ClassTemplate<char>; // WRONG
 };
```

4) 신호와 처리부는 갱신될수 없다

계승한 성원함수를 공개상태로 갱신하는 C++ 특성은 신호와 처리부를 포함하는데까지 전개되지 않는다. 여기에 옳지 않은 문법의 실례가 있다.

```
 class Whatever : public QButtonGroup {
  ...
 public slots:
  QButtonGroup::buttonPressed; // WRONG
  ...
 };
```

QButtonGroup::buttonPressed()처리부는 protected이다.

C++질문: 다중정의되는 protected성원함수를 갱신하려고 한다면 어떤 일이 생기는가?

① 모든 함수는 다중정의된다.

② 이것은 옳은 C++가 아니다.

5) 형마크로는 신호와 처리부파라메터로서 사용될수 없다

moc가 #define를 전개하지 못하므로 인수있는 형마크로들은 신호와 처리부에서 작업하지 않는다. 여기에 옳지 않은 실례가 있다.

```
 #ifdef ultrix
 #define SIGNEDNESS(a) unsigned a
 #else
 #define SIGNEDNESS(a) a
 #endif
```

```
class Whatever : public QObject {
 ...
signals:
 void someSignal( SIGNEDNESS(int) );
 ...
};
```

인수없는 #define은 기대한대로 작업한다.

6) 겹쌓인 클라스들은 신호나 처리부를 가지지 않는 신호나 처리부절에 있을수 없다

여기에 실례가 있다.

```
class A {
 Q_OBJECT
public:
 class B {
 public slots:    // WRONG
  void b();
  ...
 };
signals:
 class B {      // WRONG
  void b();
  ...
 }:
};
```

7) 구성자들은 신호나 처리부절에서 사용될수 없다

신호나 처리부절에 구성자를 왜 넣군하는가 하는것이 비결이다. 어떤 방법으로도 할수 없다. (작업과정에 그런 일이 발생하는 경우를 제외.) 구성자들이 속해있는 private, protected 혹은 public절에 그것들을 넣어야 한다. 여기에 옳지 않은 문법의 실례가 있다.

```
class SomeClass : public QObject {
 Q_OBJECT
public slots:
 SomeClass( QObject *parent, const char *name )
  : QObject( parent, name ) { } // WRONG
```

```
 ...
};
```

8) 속성들은 각각의 get와 set함수들을 포함하는 공개절앞에서 선언되여야 한다

형정의와 매개의 얻기와 설정함수들을 포함하는 public절안이나 뒤에 첫 속성을 선언하면 기대한대로 동작하지 않는다. moc는 함수들을 찾을수 없거나 형을 해결할수 없다고 통보한다. 여기에 옳지 않은 문법의 실례가 있다.

```
class SomeClass : public QObject {
 Q_OBJECT
public:
 ...
 Q_PROPERTY( Priority priority READ priority WRITE setPriority ) // WRONG
 Q_ENUMS( Priority ) // WRONG
 enum Priority { High, Low, VeryHigh, VeryLow };
 void setPriority( Priority );
 Priority priority() const;
 ...
};
```

클라스선언의 선두에 바로 Q_OBJECT뒤에서 모든 속성들을 선언함으로써 이 제한을 해결한다.

```
class SomeClass : public QObject {
 Q_OBJECT
 Q_PROPERTY( Priority priority READ priority WRITE setPriority )
 Q_ENUMS( Priority )
public:
 ...
 enum Priority { High, Low, VeryHigh, VeryLow };
 void setPriority( Priority );
 Priority priority() const;
 ...
};
```

제5절. 사용자대면부콤파일러 uic

이 절에서는 Qt GUI도구일식의 사용자대면부콤파일러를 서술한다. uic는 Qt Designer에 의해 생성된 XML로 된 사용자대면부정의파일(.ui)을 읽어들이고 대응하는 C++머리부파일 혹은 원천파일들을 창조한다. 또한 화상파일을 생성하여 C++원천코드에 생화상자료를 매몰한다.

1. 선택

1) 파일생성선택

선언을 생성한다.

uic [options] <file>

실현을 생성한다.

uic [options] -impl <headerfile> <file>

- <headerfile> - 선언파일의 이름

화상집합을 생성한다.

uic [options] -embed <project> <image1> <image2> <image3> ...

- <project> - 프로젝트이름
- <image[0..n]> - 화상파일들

편리상 uic 는 파생클라스들용의 선언 혹은 실현그루터기들을 생성할수 있다.

파생클라스선언을 생성한다.

uic [options] -subdecl <classname> <headerfile> <file>

- <classname> - 생성하려는 파생클라스의 이름
- <headerfile> - 기초클라스의 선언파일

파생클라스실현을 실현한다.

uic [options] -subimpl <classname> <headerfile> <file>

- <classname> - 생성하려는 파생클라스의 이름
- <headerfile> - 파생클라스의 선언파일

2) 일반선택

- -o file - 출력을 stdout이 아니라 'file'에 써넣는다.
- -nofwd - 생성된 머리부파일에서 전용클라스들의 앞방향선언들을 빠뜨린다. 이것은 클라스형정의가 사용된다면 필요하다.
- -tr func - 국제화에 trUtf8(sourceText, comment)가 아니라 func(sourceText, comment)를 사용한다.

2. 사용법

uic는 make에 의해서만 호출된다.

여기에 GNU make만 사용한다면 makefile을 사용할수 있다.

```
%.h: %.ui
  uic $< -o $@
%.cpp: %.ui
  uic -impl $*.h $< -o $@
```

이식가능하게 쓰려면 다음 형식의 개별적인 규칙들을 사용할수 있다.

```
NAME.h: NAME.ui
  uic $< -o $@
NAME.cpp: NAME.ui
  uic -impl $*.h $< -o $@
```

자기의 SOURCES (자기가 좋아하는 이름을 대신한다)변수에 NAME.cpp 를, 자기의 OBJECTS변수에 NAME.o을 추가하는것을 잊지 말아야 한다.

(C++원천파일을 .cpp로 이름짓는것을 좋아해도 uic는 고려하지 않으므로 .C, .cc, .CC, .cxx 혹은 자기에게 좋으면 지어는 .c++를 사용할수 있다.)

참고문헌

1. Trolltech 《Qt 3.3.6 Reference Documentation》 Trolltech, 2005.

2. C. J. Date 《An Introduction to Database Systems (7th ed.)》 ISBN, 0201385902.

이 책은 콤퓨터를 전공으로 하는 교원, 연구사, 대학생들을 위한 참고서이다.

Qt프로그람개발도구

집 필	한영철, 홍이철, 문홍남	**심 사**	김현철
편 집	차현옥	**교 정**	서금석
장 정	서경애	**콤퓨터편성**	여은정
낸 곳	교육위원회 교육정보쎈터	**인쇄소**	
인 쇄		**발 행**	

교-09-1661　　부　　값　　원